

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

**FOUNDATIONS OF NEURAL-SYMBOLIC AI:
ARCHITECTURE AND DESIGN**

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE AND ENGINEERING

by

Connor Pryor

December 2024

The Dissertation of Connor Pryor
is approved:

Professor Lise Getoor, Chair

Professor Leilani Gilpin

Professor Ian Lane

Professor Xin Eric Wang

Professor William Wang

Dean Peter Biehl
Vice Provost and Dean of Graduate Studies

Copyright © by
Connor Pryor
2024

Table of Contents

List of Figures	vii
List of Tables	ix
Abstract	xi
Dedication	xiii
Acknowledgments	xiv
1 Introduction	1
1.1 Principled Foundations of Neural-Symbolic AI	3
1.2 Contributions	4
1.3 Organization	7
I Neural-Symbolic Axioms of Integration	9
2 Neural Symbolic Architectures with Hard and Soft Constraints	10
2.1 Neural-Symbolic (NeSy) AI	11
2.2 Hard and Soft Constraints	13
2.2.1 Random Variables	14
2.2.2 Hard Constraints	15
2.2.3 Soft Constraints	17
2.2.4 Constraint Optimization	18
2.2.5 Solving Constrained Optimization Problems	20
2.3 NeSy Architectures	21
2.3.1 Symbolic as Neural Structure	21
2.3.2 Sampling Neural for Symbolic	23
2.3.3 Neural as Symbolic Parameter	26
2.3.4 Neural as Symbolic Variable	29

II	Universal Neural-Symbolic Language	33
3	Unifying NeSy through Energy-Based Models	34
3.1	Neural Symbolic Energy-Based Models as a Unifying Mathematical Framework for NeSy	35
3.2	NeSy-EBM Modeling Paradigms	37
3.2.1	Deep Symbolic Variables	37
3.2.2	Deep Symbolic Parameters	40
3.2.3	Deep Symbolic Potentials	41
3.3	Expressing NeSy Approaches via NeSy-EBMs	43
3.3.1	Semantic Loss (SL)	44
3.3.2	DeepProbLog (DPL)	47
3.3.3	Logic Tensor Networks (LTNs)	51
III	Neural-Symbolic Design Principles	54
4	Neural Symbolic Inference and Learning	55
4.1	NeSy-EBM Inference	56
4.2	NeSy Learning Design Principles	57
4.2.1	Computation Graph vs. Optimization Execution	58
4.2.2	Instance vs. Global Model Construction	60
4.2.3	Decomposed vs. Unified Task Structure	62
4.3	NeSy-EBM Learning	64
4.3.1	Definition	64
4.3.2	Learning Losses	66
4.3.3	Learning Algorithms	71
4.4	NeSy Learning Design Principles	75
4.4.1	Distant Supervision Learning	75
4.4.2	Structure-Informed Learning	77
4.4.3	Learning with Constraint Loss	78
4.4.4	Additional Design Guidelines	80
5	Challenges and Pitfalls of NeSy Modeling Paradigmns, Learning, and Reasoning	83
5.1	NeSy Modeling Paradigm Pitfalls	84
5.1.1	Unfixed Deep Symbolic Variables	84
5.1.2	Deep Symbolic Operations	86
5.2	NeSy Inference Pitfalls	87
5.2.1	Reasoning Shortcuts as Unintended Optima	87
5.2.2	Poor Factorization/Decomposition	90

5.2.3	Conditional Independence in NeSy Probabilistic Logics	92
5.3	NeSy Learning Pitfalls	93
5.3.1	Contextual Label Ambiguity	93
5.3.2	Energy Loss Degenerate Solutions	95
5.3.3	NeSy Soft Logic Pitfalls	97
IV	A General and Principled Neural-Symbolic Implementation	100
6	Deep Hinge-Loss Markov Random Fields and Neural Probabilistic Soft Logic	101
6.1	Deep Hinge-Loss Markov Random Fields	102
6.1.1	Hinge-Loss Markov Random Fields	102
6.1.2	Deep Hinge-Loss Markov Random Fields	103
6.2	Inference and Learning in Deep Hinge-Loss Markov Random Fields	105
6.2.1	MAP Inference	106
6.2.2	Learning	109
6.3	Syntax and Semantics of Neural Probabilistic Soft Logic	110
6.4	Defining NeSy-EBM Modeling Paradigms using NeuPSL	119
6.4.1	Deep Variables	119
6.4.2	Deep Weights	121
6.4.3	Deep Rules	121
6.5	NeuPSL System	123
6.5.1	System-Level Workflow	123
6.5.2	Shared Memory Mechanism	125
7	Empirical Analysis	127
7.1	Datasets and Models	129
7.2	NeSy Inference and Learning	137
7.2.1	Inference	137
7.2.2	Learning	142
7.2.3	Zero-Shot Learning	147
7.3	Comparing NeSy Approaches	148
7.3.1	Overview of NeSy Models	148
7.3.2	MNIST Addition	150
7.3.3	MNIST Addion: Overlap	152
7.3.4	Citeseer and Cora	156
7.3.5	Synthetic Mixture of Symbolic Experts	158
7.4	NeSy Pitfalls and Mitigation Strategies	162
7.4.1	Reasoning Shortcuts	163
7.4.2	Contextual Label Ambiguity	165

7.4.3	Energy Loss Degenerate Solutions:	168
7.4.4	Soft Logic Pitfalls in NeuPSL	169
8	Related Work	171
8.1	Neural-Symbolic Approaches	171
8.2	Taxonomies of NeSy Approaches	174
8.3	Energy-Based Models (EBMs)	177
9	Future Work and Limitations	180
9.1	Neural-Symbolic Axioms of Integration:	180
9.2	Universal Neural-Symbolic Language	182
9.3	Neural-Symbolic Design Principles:	183
9.4	A General Neural-Symbolic Implementation	184
10	Conclusion	186
A	Extended Model Details	200
A.1	NeuPSL Symbolic Constraints	200
A.1.1	MNIST-Add1	201
A.1.2	MNIST-Add2	202
A.1.3	Visual Sudoku	205
A.1.4	Pathfinding	207
A.1.5	Citation Network	208
A.1.6	RoadR	209
A.1.7	Zero-Shot Object Navigation	211
A.1.8	Dialog Structure Induction	211
A.1.9	Synthetic Mixture of Experts	215
A.1.10	Logic Deduction	216

List of Figures

3.1	A neural-symbolic energy-based model.	36
3.2	A deep symbolic variables model for solving a Sudoku board constructed from handwritten digits. The neural component classifies handwritten digits. Then, the symbolic component uses the digit classifications and the rules of Sudoku to fill in the empty cells.	39
3.3	A deep symbolic parameters model for citation network node classification. The symbolic component is a mixture of experts model that combines weighted arithmetic constraints. The neural component uses paper content to weigh the importance of satisfying an arithmetic constraint.	41
3.4	A deep symbolic potential model for answering questions about a set of objects' order described in natural language. The neural component is an LLM that generates syntax to create a symbolic potential. The symbolic potential is used to perform deductive reasoning and answer the question. See Example 3.2.3 for details.	42
4.1	A stochastic NeSy-EBM. The symbolic weights and the neural component parameterize stochastic policies. A sample from the policies is drawn to produce arguments of the symbolic component.	74
7.1	Average AMI for MultiWoZ, SGD Synthetic, and SGD Real (Standard Generalization, Domain Generalization, and Domain Adaptation) on three constrained few-shot settings: 1-shot, proportional 1-shot, and 3-shot.	145
7.2	Example of overlapping MNIST images in MNIST-Add1. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros.	153
7.3	Average test set accuracy and standard deviation on <i>MNIST-Add</i> datasets with varying amounts of overlap.	154
7.4	Inference and learning time for <i>MNIST-Add</i> experiments.	155
7.5	Rules, symbolic meaning, and graphical representation used to generate features and labels for the synthetic datasets.	159

7.6	Visual Sudoku Puzzle Classification Reasoning Shortcut Models.	164
7.7	Citeseer and Cora learned models with and without parameter simplex constraints.	169
A.1	NeuPSL MNIST-Add1 Symbolic Model	201
A.2	NeuPSL MNIST-Add2 Symbolic Model	203
A.3	NeuPSL Visual Sudoku Symbolic Model	206
A.4	NeuPSL Pathfinding Symbolic Model	207
A.5	NeuPSL Citation Network Symbolic Model	208
A.6	NeuPSL RoadR Object Detection Symbolic Model	209
A.7	NeuPSL Zero Shot Object Navigation Symbolic Model	210
A.8	NeuPSL SGD Dialog Structure Induction Symbolic Model	212
A.9	NeuPSL MultiWoZ Dialog Structure Induction Symbolic Model	213
A.10	NeuPSL Synthetic Mixture of Experts Symbolic Model	216
A.11	NeuPSL Logic Deduction Prompt for Generating Symbolic Model	217

List of Tables

2.1	Fuzzy/Soft Logic Constraints for NEGATION, AND, and OR.	29
7.1	Digit accuracy and constraint satisfaction consistency of the ResNet18 and NeuPSL models on the MNIST-Add- k and Visual-Sudoku datasets.	139
7.2	Accuracy of finding a minimum cost path (Min. Cost Acc.) and consistency in satisfying continuity constraints (Continuity) of the ResNet18 and NeuPSL models on the Pathfinding dataset.	139
7.3	Object detection F1 and constraint satisfaction consistency of the DETR and NeuPSL models on the RoadR dataset.	139
7.4	Node classification accuracy of the SGC and NeuPSL models on the Citeseer and Cora datasets.	140
7.5	Comparison of accuracy in answering logical deduction questions using two large language models, GPT-3.5-turbo and GPT-4 [98], across three methods: Standard, Chain of Thought (CoT), and NeuPSL.	141
7.6	Comparison of success rate (SR) and success rate weighted by inverse path length (SPL) in zero-shot object navigation on MP3D [22], HM3D [106], and RoboTHOR [40] benchmarks, across three methods: CLIP on Wheels (CoW) [54], ZSON [79], and NeuPSL (ESC) [145]	141
7.7	Test set accuracy and standard deviation on MNIST-Add experiments. Results reported here are run and averaged over ten splits.	143
7.8	Digit accuracy of the ResNet18 models trained with varying levels of supervision.	146
7.9	Test set performance on all datasets. All reported results are averaged over 10 splits. The highest-performing methods per dataset and learning setting are bolded. A random baseline has AMI zero and class-balanced accuracy equal to inverse class size.	147
7.10	Test set accuracy and standard deviation on <i>MNIST-Add</i> . Results reported here are averaged over the same ten splits. Best results and those within one standard deviation of the best are in bold.	150

7.11	Test set accuracy and inference runtime in seconds on two citation network datasets.	157
7.12	Average categorical accuracy on the highest correlation between features and labels for $OH + OH$, $G + OH$, and $G + G$ data settings. Best-performing methods are in bold.	160
7.13	Average categorical accuracy on varying covariance matrices used for synthetic data generation in the $G + OH$ and $G + G$ data settings. Higher covariance results in a lower correlation between features and labels. Best-performing methods are in bold.	161
7.14	Test set final objective and puzzle and digit accuracy for NeuPSL models on visual sudoku puzzle classification with and without a reasoning shortcut mitigation.	165
7.15	Test set digit accuracy predicted by the neural and symbolic components for NeuPSL models on visual sudoku with and without ambiguous local context.	167
7.16	Test set accuracy for the NeuPSL models with and without the zeroed weights degenerate solution.	168
7.17	Test set accuracy for the NeuPSL models with and without a pretrained backbone.	170

Abstract

Foundations of Neural-Symbolic AI:

Architecture and Design

by

Connor Pryor

Deep neural networks have become synonymous with artificial intelligence, playing a crucial role across industry, academia, and everyday life. Despite their impressive capabilities, these models still exhibit fundamental limitations, including perpetuating human bias, a lack of robust prediction guarantees, and unreliable explanations. In response to these limitations, the past decade has seen a revival of symbolic approaches integrated with the data-driven strengths of neural networks, resulting in a broad array of neural-symbolic (NeSy) methods [16, 32, 33, 87]. While promising, the field of neural-symbolic AI is still in its early stages, with many current methods conflating the distinct processes of inference, learning, and architectural design. This lack of separation makes it difficult to compare and evaluate the effectiveness of different approaches across various tasks. NeSy AI needs to establish a principled foundation that (1) provides the axioms of neural-symbolic integration, (2) defines a universal neural-symbolic language, (3) categorizes neural-symbolic design principles, and (4) collects a set of general and principled implementations. In this dissertation, I aim to develop a strong foundation for NeSy AI, starting with clear architectural axioms for integrating symbolic and subsymbolic components framed through hard and soft constraints.

My contributions are fivefold: (1) I address the conflation of inference, learning, and the neural-symbolic interface by categorizing approaches through key architectural axioms, providing a clear base for NeSy research. (2) I formalize these architectural choices through a unifying mathematical framework, enabling the definition and comparison of most NeSy approaches. (3) Leveraging this formalization, I identify effective learning strategies and common pitfalls that impact a wide range of NeSy approaches, offering actionable insights for improving their design and performance. (4) Based on these insights,

I develop a novel, practical NeSy implementation that supports most architectural choices and learning strategies. (5) I validate this implementation across multiple domains, including graph node labeling, image classification, autonomous event detection with safety requirements, complex natural language question answering, and dialog structure induction. These contributions bring neural-symbolic AI closer to a unified foundation by providing the terminology, mathematical tools, and design principles necessary to create scalable, interpretable, and adaptable systems that effectively integrate neural and symbolic reasoning.

*To my family for their patience,
my friends for their encouragement,
and my colleagues for their support.*

Acknowledgments

The work of a Ph.D. student is shaped, first and foremost, by their advisor, alongside the support of friends, colleagues, and family. My journey has been no exception. I would like to begin by expressing my deepest gratitude to my advisor, Lise Getoor, for her unwavering guidance, support, sacrifices, and her remarkable ability to inspire curiosity in research. Lise has profoundly influenced both my growth as a researcher and my passion for artificial intelligence and machine learning. My Ph.D. journey began in her Introduction to AI course, where my Pac-Man agent won the Capture the Flag competition. While I once believed this achievement might have played a role in securing a place in her lab, I now realize that she values passion, dedication, and hard work above all else. Throughout my Ph.D., Lise has been instrumental in shaping my approach to research. She taught me how to craft high-quality papers and consistently emphasized the importance of grounding research in rigorous results and sound theory. Her mentorship extended far beyond technical guidance, providing a constant source of encouragement, support, and motivation. I am profoundly grateful for everything she has done to help me succeed, and I am excited about the opportunities that lie ahead. While I will continue striving to be less pessimistic about research, I suspect my tendency to ask tough questions—something Lise has always modeled—will remain a defining trait of both me and my research approach.

Proper research is never done in a vacuum; it thrives through collaboration and the support of a dedicated community. I am deeply grateful to all the collaborators I have had the privilege of working with over the years. I was particularly fortunate to work alongside Eriq Augustine, one of the most brilliant coding researchers I have ever met. His technical expertise and relentless commitment to excellence had an immeasurable impact on both the lab’s work and my personal development. Reflecting on my time in the lab, it’s clear how profoundly Eriq influenced my coding style, efficiency, and attention to detail—especially through his “encouragement” to follow good coding practices via his advocacy of Git version control. His impact on me extended far beyond technical skills; Eriq has been an exceptional colleague, a valued mentor, and a dear friend. Alongside Eriq, I had the privilege of working with Charles Dickens, a brilliant theoretical researcher whose work has pushed the boundaries of optimization in neural-symbolic systems. Some

of my most cherished moments during my Ph.D. stem from the countless hours we spent at the whiteboard, where we “solved” NeSy (at least in theory), deciphered cryptic approaches in the field, and debated the broader implications of our research. Charles and I faced the challenges of the Ph.D. journey together, and he has become not only a trusted collaborator but also a close friend. Some of my fondest memories during my Ph.D. are of the times I spent with both Eriq and Charles—not necessarily doing active research but brainstorming ideas over drinks or playing billiards. I would also like to extend special thanks to Yatong Chen, Varun Embar, and Sriram Srinivasan. While our collaboration on research was limited, our conversations about life, the future, and myriad other topics provided invaluable perspective and support throughout this journey. Further, I want to express my gratitude to the entire LINQS Lab. I feel extraordinarily lucky to have been part of such a kind, supportive, and intellectually stimulating group of people. The community fostered within the lab has been instrumental in shaping both my research and my growth as a person. Finally, I would like to extend my thanks to the many great researchers I have met throughout my research journey: Alon Albalak, Samantha Andrzejczek, Samy Badreddine, Tania Bedrax-Weiss, Shresta B.S., William Cohen, Artur d’Avila Garcez, Kai-Wei Chang, Johnnie Chang, Wenhui Chen, Alexandra DiGiacomo, Shobeir Fakhraei, Golnoosh Farnadi, Changyu Gao, Eleonora Giunchiglia, Vihang Godbole, Frank van Harmelen, Pegah Jandaghi, Dhawal Joharapurkar, Seyed Mehran Kazemi, Pigi Kouki, Emile van Krieken, Fabrice Kurmann, Caleb Levy, Jeremiah Zhe Liu, Jaron Maene, Alex Miller, Jay Pujara, Luc de Raedt, Deepak Ramachandran, Rishika Singh, Lennert de Smet, Dhanya Sridhar, Niharika Srivastav, Andrew Thach, Jason Ting, Sabina Tomkins, Yi-Lin Tuan, Guy Van den Broeck, Vibin Vijay, Stephen Wright, Quan Yuan, Elena Zheleva, Kaiwen Zhou, and many others.

While research is never done in a vacuum, it has a way of consuming one’s time, energy, and mental state. I am endlessly grateful to my incredible fiancée, Maria, my best friend, Dawson, and my cat, Ashton, for ensuring it didn’t consume me completely. Their constant patience, encouragement, and support over the past six years have been the foundation that kept me grounded. Maria, it has been an amazing journey growing together as a couple while we both build our careers. She may not realize just how much

I notice and appreciate everything she does—not just the big things, like standing by me through challenging times, but also the countless little things: reminding me to eat when I would otherwise skip meals, pulling me out of a research-induced haze with a silly joke, encouraging me to pursue neglected hobbies, and so much more. I am so proud of all she has accomplished in her career and life, and I am beyond excited for the future we are building together. And to think—we even tried to get married three days before my defense! Dawson has been a source of constant encouragement and a great listener, always willing to hear me ramble about research with genuine interest and thoughtful advice. His empathy and support have helped me keep my cool, and his insights often encouraged me to think about the broader impact I want to have on my life and the lives of others. Watching him complete his graduate studies and secure a job in his field was both inspiring and reassuring during times when I felt uncertain about my own career. I am so proud of all he has accomplished and look forward to exploring new hobbies and projects with him now that I have more time. Outside of my Ph.D., the most meaningful milestones in my life have been shared with these two—buying and renovating a home, hosting events, planning weddings, and building our careers. I couldn’t have asked for a better duo to share this journey with, and I am incredibly grateful for their presence in my life.

Research can never be accomplished without taking breaks and relaxing the mind. If it had not been for all of my friends and family’s support during the times I wasn’t working, research would have stagnated. Special thanks to my parents, Anna and Mark, who have been the kindest and most patient people a Ph.D. student could ask for. The foundation they built—instilling strong morals, a deep curiosity for knowledge, and a sense of perseverance—has guided me through this Ph.D. Furthermore, their understanding of the time-consuming nature of a Ph.D., along with their acceptance of my occasional absence, has been immeasurably important to the success of this journey. Additionally, thanks to my brother, Austin, he has always had a way of pulling me out of my bubble. Some of the fondest memories of my Ph.D. come from the time spent with him—whether watching movies, rocking out at concerts, attending events, or venting about work. Those moments provided both therapy and joy. Thank you to the rest of my family: Dolores, Christine, Kevin, Greg, Judy, Taylor, Carly, Dorothy, Lisa, Alessandra, Gianni, and Jim, as well as

all the members of my extended family. Finally, I want to give special thanks to a few friends: Jon, Andrew, Huimee, Marc, Justin, Lauren, Connor, Luz, Jonatan, Mary, Shir, Shay, Reem, Thomas, Sylvester, Zoe, Jay, Mark, Karen, Charlie, and Peter.

Lastly, I would like to extend my gratitude to my dissertation reading committee: Dr. Lise Getoor, Dr. Leilani Gilpin, Dr. Ian Lane, Dr. Xin Eric Wang, and Dr. William Wang. Your valuable time, insightful feedback, and thoughtful suggestions have significantly enhanced the quality and depth of my dissertation. Thank you for your support and guidance throughout this process.

Portions of this dissertation were supported by National Science Foundation grants CCF-2023495; and an unrestricted gift from Google. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

Chapter 1

Introduction

In recent years, in the eyes of mainstream media and many researchers, deep neural networks have become the cornerstone of artificial intelligence, achieving state-of-the-art performance across various domains, such as natural language processing and computer vision. Despite the impressive achievements, researchers have identified several critical limitations. Neural models typically rely on vast amounts of high-quality training data, operate opaquely in their decision-making processes, and struggle with spurious correlations rather than learning accurate generalizations. Additionally, their black-box nature makes them unsuitable for fields where interpretability and reasoning are crucial, such as healthcare, autonomous driving, and legal decision-making. These issues raise concerns about the safety, trustworthiness, and ethical implications of deploying these types of machine learning models in real-world, high-stakes environments.

In an effort to overcome these shortcomings, there has been a growing interest in methods combining neural models' pattern recognition capabilities with structured symbolic reasoning approaches, which are well-studied and have a long history in AI. Integrating neural networks with symbolic reasoning is often referred to as neural-symbolic (NeSy) AI [16, 32, 33, 55, 87]. It seeks to leverage the strengths of both paradigms: neural networks' adaptability and data-driven learning and symbolic systems' logical, interpretable reasoning. This integration holds promise for addressing some of the inherent weaknesses of purely subsymbolic approaches by ensuring that predictions are not only accurate but

also logically sound and interpretable.

The potential benefits of neural-symbolic integration are widely recognized, yet the field remains nascent. Despite this, it draws upon a rich history of research of separate symbolic and subsymbolic learning and reasoning, spanning decades. One of the primary challenges, however, lies in the lack of clarity regarding the “correct” method for integrating these paradigms. This ambiguity has given rise to a diverse array of NeSy methods, systems, and loss functions [3, 4, 5, 6, 7, 8, 13, 14, 27, 28, 30, 31, 36, 42, 46, 47, 49, 57, 64, 65, 78, 81, 85, 86, 88, 89, 90, 94, 99, 103, 108, 111, 114, 116, 117, 119, 122, 125, 126, 128, 132, 136, 138, 140, 141]. Further, many of these approaches conflate the distinct processes of reasoning, learning, and architectural design within neural-symbolic integration, leading to a lack of conceptual clarity. As a result, systems are often *ad hoc*, tailored to specific problems, and challenging to generalize. Furthermore, approaches rooted in similar theoretical foundations are frequently presented as novel, differentiated only by syntactic variations. This fragmentation complicates efforts to compare, evaluate, and synthesize insights across the field.

To address the ambiguity, conflation, and sheer volume of approaches in neural-symbolic integration, there has been a concerted effort within the community to develop taxonomies that group and organize these systems. These taxonomies span various dimensions, including the representation of symbolic knowledge [16, 70, 87, 127], the interaction between neural and symbolic components [12, 16, 32, 34, 37, 43, 44, 87, 15], learning and reasoning [32, 43, 45, 87, 15], application domains [14, 17, 43, 81, 87, 142], common shortcuts [82, 83, 84], and system languages [72]. They also bridge connections to related fields such as statistical relational learning [87], graph neural networks [73], and knowledge graphs [143]. While these taxonomies provide valuable structure and insight, they address only the symptoms of a larger underlying challenge: the need for principled foundations for neural-symbolic AI. Such foundations are essential for unifying the field, fostering communication within and beyond NeSy, and preventing the redundant reinvention of ideas.

1.1 Principled Foundations of Neural-Symbolic AI

Despite the rapid expansion of the NeSy field, a principled foundation for neural-symbolic AI remains a critical necessity. I propose four key milestones to establish such a foundation: (1) *axioms of neural-symbolic integration*, (2) *universal neural-symbolic language*, (3) *neural-symbolic design principles*, and (4) *general and principled implementations*. Each milestone builds upon the previous one, representing a cohesive pathway toward a robust and unified foundation for neural-symbolic AI that is more accessible to the broader machine learning community and new members.

Neural-Symbolic Integration Axioms: Neural-symbolic AI must first establish the foundational axioms of integration. Rather than beginning through the design of neural-symbolic systems, it is crucial to begin with how neural and symbolic components can effectively integrate. This step is pivotal because NeSy AI benefits from decades of research in standalone neural and symbolic paradigms, including theory, inference, learning, systems, and applications. By establishing principled integration axioms, the field can build on these rich foundations in a systematic and structured way. At its core, NeSy AI requires a motivating and intuitive starting point, allowing the formalism to evolve from this foundational base naturally.

Universal Neural-Symbolic Language: Building upon the axioms of NeSy integration, the next critical milestone is establishing a universal neural-symbolic language capable of formally representing these axioms. This language would serve as a unifying framework for articulating integration principles, enhancing clarity and precision within the NeSy community, and facilitating effective communication across the broader machine-learning field. Moreover, this addresses a persistent challenge in the field: approaches rooted in similar theoretical foundations often appear novel due to differences in syntax. Standardizing representation mitigates redundancy, minimize the need for papers to repeatedly restate foundational concepts, and allow researchers to concentrate their efforts on advancing the field.

Neural-Symbolic Design Principles: With a universal language for describing NeSy AI systems in place, the next step is to define and organize NeSy design principles. These principles encompass three key areas: (1) *losses and algorithms* that drive the reasoning and learning processes, (2) *implementation strategies and motivating tasks* that inform practical applications and system design, and (3) *common pitfalls and mitigation strategies* to address challenges often encountered in neural-symbolic integration. Together, these design principles form a practical guideline for effectively designing novel neural-symbolic approaches and applications.

General and Principled Implementations: The final milestone is the development of general and expressive NeSy frameworks that build upon the foundations established by the previous milestones. These frameworks should be grounded in the theoretical representation and integration axioms, ensuring consistency and coherence across implementations. Within the NeSy community, reaching a consensus on which principled approaches should serve as foundational is essential. Future work can then either align with these foundational frameworks, identifying itself as a variation, or demonstrate sufficient distinction to justify further implementation. This alignment would enable shared design principles, such as standardized losses and algorithms for learning processes, and facilitate addressing challenges through common mitigation strategies. Ultimately, principled implementations will streamline research efforts, enhance comparability, and accelerate progress in advancing NeSy systems.

1.2 Contributions

The aspects of my research are aligned with the foundational aspect required for principled neural-symbolic AI. Below, I provide a brief overview of these contributions, with further specific contributions detailed in subsequent chapters.

Neural-Symbolic Integration Axioms: While neural-symbolic research consistently utilizes foundational neural-symbolic integration techniques, I argue that the community

lacks a cohesive and explicitly defined set of axioms for integration. To address this, motivated by current work [32, 44, 87], I take a first step in defining four key architectural axioms that are current cornerstones for NeSy research: (1) *Symbolic as Neural Structure*: the neural architecture is defined by a symbolic model’s structure, where the system remains subsymbolic and learns the relationships through data. (2) *Sampling Neural for Symbolic*: the neural architecture performs a non-differentiable sampling process that can represent random variables, parameters, or structures within a symbolic model. (3) *Neural as Symbolic Variable*: the neural architecture outputs are treated as differentiable random variables within a symbolic structure, enabling direct influence over the variables used in the prediction program. (4) *Neural as Symbolic Parameter*: the neural architecture outputs are treated as differentiable parameters within a symbolic structure, allowing the neural model to exert indirect influence on random variables or interact with other components of the symbolic model—such as parameters associated with individual constraints. In addition to these four key architectural axioms, given the community’s current emphasis on logic-based NeSy approaches, I provide detailed examples for each architectural axiom grounded in symbolic logic, offering familiar and tangible scenarios to elucidate these principles. Expanding and refining this set of axioms is one of the most critical next steps for advancing and solidifying the foundations of NeSy research.

Universal Neural-Symbolic Language: With the axioms of integration specified as architectural choices, the next step is to define a universal language to formalize and foster communication within and outside the NeSy community. While recent work has focused on creating universal syntaxes for logic-based systems [72], a universal mathematical framework is still needed to unify the practical implementations of NeSy approaches. To facilitate that need, I introduce Neural-Symbolic Energy-Based Models (NeSy-EBMs) [44, 45, 103], as that unifying mathematical framework which combines neural and symbolic components through energy functions. The neural component can represent a wide range of neural architecture, from simple multi-layer perceptrons to more complex models, such as large language models, while the symbolic component can encompass complex optimization processes. However, a general definition alone is insufficient to formalize the architectural

axioms introduced earlier. To bridge this gap, I present three critical neural-symbolic modeling paradigms[44]—*deep symbolic variables* (DSVar), *deep symbolic parameters* (DSPar), and *deep symbolic potentials* (DSPot)—which serve as the theoretical building blocks for constructing systems that embody the architectural axioms. Finally, to illustrate the practicality and unifying power of NeSy-EBMs, I demonstrate how several prominent NeSy approaches can be naturally expressed within this framework [103]. This not only highlights the framework’s flexibility but also reveals deeper connections and insights into the theoretical and practical formulations of existing methods.

Neural-Symbolic Design Principles: With the establishment of NeSy-EBMs as a universal mathematical framework, the next step toward a robust foundation for NeSy involves categorizing and defining a comprehensive set of design principles commonly encountered in neural-symbolic systems. These principles serve as a structured roadmap for developing and implementing practical NeSy approaches. For this milestone, I present inference and learning techniques defined with NeSy-EBMs [45], applicable to systems that align with this framework and conform to specific assumptions. Through this, I detail five inference formulations addressing common tasks and outline four learning techniques [45], including one for separate neural and symbolic learning and three for joint parameter learning. While much of the formal proofs and theory are contributions of my collaborator Charles, they are presented here to offer a comprehensive view of NeSy design principles. For further details, I direct readers to his seminal work [43]. In addition to tasks and techniques, I emphasize three critical reasoning design decisions: *computation graph versus optimization-based execution*, *instance versus global model construction*, and *decomposed versus unified task structures*. Furthermore, on the learning side, I categorize a collection of training design patterns and guidelines: *distant supervision learning*, *structure-informed learning*, *learning with constraint loss*, *practical NeSy learning tasks*, *pre-training strategies*, and *post-training use-cases*. Finally, I present a set of prevalent pitfalls: *pitfalls in modeling paradigms*, *pitfalls in inference*, and *pitfalls in learning*.

General and Principled Implementations: Finally, I advance the development of general and principled NeSy implementations by introducing Neural Probabilistic Soft Logic (NeuPSL) [103], a highly expressive and efficient framework for constructing NeSy-EBMs built on the well-studied probabilistic programming language Probabilistic Soft Logic [11]. To establish NeuPSL as a principled framework, I begin by extending the underlying undirected graphical representation of hinge-loss Markov random fields to *deep hinge-loss Markov random fields*. Building on this, I define three distinct NeuPSL formulations aligned with the DSVar, DSPar, and DSPot modeling paradigms [44], ensuring alignment with the architectural axioms and theoretical constructs outlined earlier. To demonstrate NeuPSL’s generality and versatility, I showcase its practical applicability across nine diverse data settings, tackling real-world challenges such as event detection in autonomous vehicles, dialog structure induction [104], pathfinding, and logic-driven question answering [44].

1.3 Organization

- **Part I: Neural-Symbolic Axioms of Integration** provides the foundational background on neural-symbolic methods, defines the concepts of hard and soft constraints, and introduces the four key architectural axioms (Chapter 2).
- **Part II: Universal Neural-Symbolic Language** formally defines Neural-Symbolic Energy-Based Models (NeSy-EBMs), outlines the associated modeling paradigms, and reformulates prominent NeSy approaches within this language (Chapter 3).
- **Part III: Neural-Symbolic Design Principles** formalizes NeSy-EBM inference and learning by detailing algorithms, losses, and strategies (Chapter 4) and categorizes a set of prominent NeSy pitfalls, offering insights into their causes and mitigations (Chapter 5).
- **Part IV: Implementation and Empirical Analysis** introduces *NeuPSL*, a practical method for implementing a collection of NeSy-EBM modeling paradigms and architectural axioms (Chapter 6), and presents an extensive empirical evaluation,

including a study of design choices, a comparison of NeSy approaches, and demonstrations of pitfalls and learning strategies (Chapter 7).

- Finally, Chapter 8 presents related work, Chapter 9 discusses limitations and future directions for neural-symbolic research, and Chapter 10 concludes.

Part I

Neural-Symbolic Axioms of
Integration

Chapter 2

Neural Symbolic Architectures with Hard and Soft Constraints

With decades of research dedicated to separate neural and symbolic theories [33, 34, 32], I argue that the foundation of neural-symbolic AI should not begin with the development of individual syntax or theories. Instead, it should start with axioms of neural-symbolic integration, i.e., principles that define how neural and symbolic components should interact. Establishing these axioms is fundamental, as they provide a solid base from which NeSy methods can draw on the well-established theories already developed. For instance, the subsequent chapter (Chapter 3) builds upon the Energy-Based Model theory [74] to formalize a universal mathematical language for NeSy. Therefore, in this chapter, I propose a set of *architectural axioms*, which establish the foundational principles for how subsymbolic and symbolic components interface. These axioms are designed to be general and modular, ensuring they act as the foundation for a plethora of NeSy approaches and provide the groundwork for future exploration and integration.

The organization of this chapter is as follows: Section 2.1 offers a brief history of NeSy AI, outlines the expansive scope of the field, and introduces a running example that will serve as a reference throughout the chapter. Section 2.2 presents the necessary background and notation, including the formalization of random variables, hard and soft constraints, and methods for constrained optimization. Building on this foundation, Sec-

tion 2.3 introduces the four proposed architectural axioms, complete with detailed examples rooted in logic-based NeSy approaches, providing a concrete and familiar framework for practical implementation.

2.1 Neural-Symbolic (NeSy) AI

The promise of integrating neural and symbolic methods to leverage their complementary strengths has been a driving force in artificial intelligence (AI) research for decades. Neural-symbolic (NeSy) AI, as a field, has emerged from this aspiration, growing steadily over the past two decades with regular workshops since 2005 [96] and its first dedicated conference in 2024 [97]. At its core, NeSy research aims to develop algorithms and architectures that seamlessly combine neural learning and symbolic reasoning [7, 14, 28, 32, 33, 34, 81, 103, 138, 141].

The origins of NeSy AI can be traced back to the resurgence of neural networks during the 1980s and 1990s, a period marked by debates about the relative merits of symbolic versus neural approaches to AI. Patrick Winston (1991) captured this dichotomy and the potential for synthesis in his reflection:

“Today, some researchers who seek a simple, compact explanation hope that systems modeled on neural nets or some other connectionist idea will quickly overtake more traditional systems based on symbol manipulation. Others believe that symbol manipulation, with a history that goes back millennia, remains the only viable approach. ... Instead, ... AI must use many approaches. AI is not like circuit theory and electromagnetism. There is nothing wonderfully unifying like Kirchhoff’s laws are to circuit theory or Maxwell’s equations are to electromagnetism. Instead of looking for a ‘right way,’ the time has come to build systems out of diverse components, some connectionist and some symbolic, each with its own diverse justification.”[93, page 35]

While the NeSy community has made significant strides in recent years, much of the progress has been fragmented, driven by diverse goals and assumptions rather than a unified foundation. This lack of cohesion has resulted in a proliferation of methods, many of which lack clear definitions or shared principles. Consequently, NeSy research today represents a patchwork of principled approaches, often created from some well-established

system such as ProbLog. Furthermore, by deriving NeSy from a systems approach it has become challenging to formalize what defines a neural-symbolic method. For example, some argue that a NeSy method must incorporate symbolic knowledge with formal syntax and semantics akin to logical systems, while others extend the definition to include more flexible syntaxes, such as natural language. This thesis adopts a broad but structured perspective on NeSy systems:

A neural-symbolic method incorporates human-designed knowledge into a model’s architecture, data processing, or predictions.

For instance, even the process of averaging information in a Graph Neural Network (GNN) [111] can be seen as a symbolic operation at a fundamental level, as it reflects a rule-based aggregation of information from graph nodes. This view extends to other neural networks, such as Convolutional Neural Networks (CNNs) [75] and Attention Networks [129], which incorporate symbolic components through their inductive biases. In CNNs, the inductive bias assumes spatial locality in data, and attention networks employ an inductive bias that emphasizes relationships between all input elements. At its core, this suggests that only pure, one-layer dense networks lack such explicit symbolic structures, relying purely on data-driven learning without incorporating human-designed biases.

Given this generously broad interpretation, NeSy systems could, in principle, encompass a significant portion of the neural network literature. However, such a broad definition risks undermining the concept’s utility by making it overly inclusive. To maintain clarity and focus, this work narrows its scope to architectural choices that emphasize a clear separation between subsymbolic and symbolic components, even as the architectural axioms introduced later in this chapter provide the flexibility to accommodate symbolic inductive biases within neural methods. With this foundation established, the following introduces a canonical neural-symbolic task that has become a standard reference within the community.

Example 2.1.1. *MNIST-Addition [81] is a canonical neural-symbolic problem that involves predicting the sum of two digits, each represented as an MNIST image. In this setup, the neural component, $g_{w_{nn}}$, parameterized by $w_{nn} \in \mathcal{W}^{nn}$, serves as a digit classifier. It takes*

an MNIST image as input and outputs a distribution over possible digits, i.e., $\{0, \dots, 9\}$. The symbolic component is an addition constraint, $C(\cdot, \cdot)$. Inference/prediction in this context involves first passing the images through the digit classifier to obtain a distribution, passing this information to the symbolic component, and then solving the sum.

Note 2.1.1. The definition of the neural-symbolic system depends on how the neural outputs are integrated into the symbolic constraint, how the constraint itself is formulated, and the specific inference process used to solve it.

For example suppose the addition ($\mathbf{3} + \mathbf{5} = 8$). The images $\mathbf{3}$ and $\mathbf{5}$ are given to the neural model $g_{w_{nn}}$ to produce an output:

$$\begin{aligned} g_{w_{nn}}(\mathbf{3}) &= \{P(0)_1 = 0.0, P(1)_1 = 0.0, P(2)_1 = 0.0, P(3)_1 = 0.9, P(4)_1 = 0.0, \\ &\quad P(5)_1 = 0.0, P(6)_1 = 0.0, P(7)_1 = 0.0, P(8)_1 = 0.1, P(9)_1 = 0.0\}, \\ g_{w_{nn}}(\mathbf{5}) &= \{P(0)_2 = 0.0, P(1)_2 = 0.0, P(2)_2 = 0.0, P(3)_2 = 0.0, P(4)_2 = 0.0, \\ &\quad P(5)_2 = 0.8, P(6)_2 = 0.0, P(7)_2 = 0.0, P(8)_2 = 0.0, P(9)_2 = 0.2\}. \end{aligned}$$

Then to perform the addition, suppose the $\arg\max$ is taken such that the addition is in integer values:

$$C(g_{w_{nn}}(\mathbf{3}), g_{w_{nn}}(\mathbf{5})) = \arg\max(g_{w_{nn}}(\mathbf{3})) + \arg\max(g_{w_{nn}}(\mathbf{5})) = 3 + 5 = 8$$

Note 2.1.2. Learning in this setting is influenced by several factors, such as the type of ground truth data available, the differentiability of the symbolic component, and the method used to integrate the neural distribution into the symbolic reasoning process. A common approach involves using the ground truth sum within a differentiable NeSy system, allowing gradients to propagate from the symbolic component to the subsymbolic component. Examples of learning processes applied to this scenario will be discussed in Section 2.3.

2.2 Hard and Soft Constraints

This section presents the essential notation used throughout the remainder of this dissertation, covering the concepts of *random variables*, *hard and soft constraints*, and

constrained optimization. Readers already familiar with these foundational concepts may wish to proceed directly to the subsequent section on NeSy architectures (Section 2.3).

2.2.1 Random Variables

This dissertation explores various tasks, problems, and representations that require reasoning about uncertainty and handling different types of *events*. In the context of probabilistic models, these events are represented using *random variables*, which provide a formal framework for capturing the uncertainty inherent in various outcomes. This section will summarize and formalize random variables as presented in Koller and Friedman (2009), which will be applied consistently throughout the dissertation.

A *random variable* is a formal way of representing an attribute or property of an outcome [69]. For example, the random variable *Grade* may represent a person’s grade in a class, which could take on values such as $\{A, B, C, D, \text{ or } F\}$ for a discrete case, or a continuous value like 95.5% in a percentage-based grading system. Formally, a random variable is a measurable function $f(\cdot)$ that maps outcomes from a sample space Ω to a set of possible values. The set of values that a random variable X can take is denoted as $\text{Val}(X)$. Random variables are typically represented by uppercase Roman letters (e.g., X, Y, Z), while lowercase Roman letters (e.g., x, y, z) denote specific values that these random variables take.

The models described in this dissertation employ three main categories of random variables: *observed random variables*, *target random variables*, and *latent random variables*:

- **Observed Random Variables:** *Observed random variables* represent values that are directly known or measured from empirical data. Let $\mathbf{X} = \{X_1, X_2, \dots, X_n\}$ denote the set of observed random variables. Denote \mathbf{x} as the value for a specific observed instance, where $\mathbf{x} \in \text{Val}(\mathbf{X})$. For example, X_i might represent an individual’s height, where the actual measurement (e.g., $x_i = 180$ cm) is known.
- **Target Random Variables:** *Target random variables* are the quantities the models aim to predict or infer. Let $\mathbf{Y} = \{Y_1, Y_2, \dots, Y_m\}$ represent the set of target random

variables. Denote \mathbf{y} as the value for a specific observed instance, where $\mathbf{y} \in \text{Val}(\mathbf{Y})$. For instance, in a classification task, \mathbf{Y} could represent the predicted label for an image, such as “cat” or “dog,” and \mathbf{y} would denote the specific predicted value.

- **Latent Random Variables:** *Latent random variables* represent hidden or unobserved factors influencing the observed and target variables. Let $\mathbf{Z} = \{Z_1, Z_2, \dots, Z_l\}$ represent the set of latent random variables. Denote \mathbf{z} as the value for a specific observed instance, where $\mathbf{z} \in \text{Val}(\mathbf{Z})$. For example, in a mixture model, Z_i might represent the latent class or cluster to which an observation belongs.

When discussing categorical random variables, denoted x_1, x_2, \dots, x_k for enumeration. For example, the following expression shows that the probabilities sum to 1:

$$\sum_{i=1}^k P(X = x_i) = 1$$

The explicit notation $P(X = x)$ is typically shorthand as $P(x)$ when the random variable is clear from context. Another useful shorthand is $\sum_{\mathbf{x}} P(\mathbf{x})$, which refers to summing over all values. For example, the previous summation can be written as:

$$\sum_x P(\mathbf{x}) = 1.$$

2.2.2 Hard Constraints

A hard constraint C_h defines a condition over the values of target variables $\mathbf{y} = \{y_1, y_2, \dots, y_m\}$ and, optionally, observed variables $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ or latent variables $\mathbf{z} = \{z_1, z_2, \dots, z_p\}$.¹ The constraint evaluates to 1 if the condition is satisfied and 0 otherwise:

$$C_h(\mathbf{y}) = \begin{cases} 1 & \text{if the constraint is satisfied,} \\ 0 & \text{if the constraint is violated.} \end{cases}$$

Hard constraints are crucial in neural-symbolic systems, as they typically ensure that logical rules and structural relationships between variables are maintained. Depending on the

¹All definitions in this section can be straightforwardly extended to include either observed or latent variables.

problem context, these constraints may take different forms: equality, inequality, integer, and more complex structural constraints.

Equality Constraints: An equality constraint enforces that a function over the values of the random variables must equal some constant c . Formally, an equality constraint C_{equality} is expressed as:

$$C_{\text{equality}}(\mathbf{y}) = \mathbb{I}[f(\mathbf{y}) = c],$$

where $f(\mathbf{y})$ is a function of the target values, c is a constant, and \mathbb{I} is the indicator function that returns 1 if the condition is true and 0 otherwise.

Example 2.2.1. *In a sports tournament, only one team can win. Let $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ represent the values of the target variables, where $y_i = 1$ if team i wins and $y_i = 0$ otherwise. The equality constraint ensures exactly one team wins:*

$$C_{\text{equality}}(\mathbf{y}) = \mathbb{I}\left[\sum_{i=1}^n y_i = 1\right].$$

Inequality Constraints: Inequality constraints enforce that a function over the random variable values must lie within certain bounds, either as a lower or upper bound. Formally, an inequality constraint $C_{\text{inequality}}$ is written as:

$$C_{\text{inequality}}(\mathbf{y}) = \mathbb{I}[f(\mathbf{y}) \leq c] \quad \text{or} \quad \mathbb{I}[f(\mathbf{y}) \geq c],$$

where $f(\mathbf{y})$ is a function of the target values, c is a constant, and \mathbb{I} is the indicator function that returns 1 if the condition is true and 0 otherwise.

Example 2.2.2. *Suppose a system is tasked with allocating resources to different departments, and the total budget must not exceed B . Let $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ represent the values of resources allocated to each department. The inequality constraint ensures the total allocation does not exceed B :*

$$C_{\text{inequality}}(\mathbf{y}) = \mathbb{I}\left[\sum_{i=1}^n y_i \leq B\right].$$

Integer Constraints: In some cases, the values of the target variables must be integers (e.g., the number of tasks assigned to workers). These constraints are essential when only discrete values are valid. Formally, an integer constraint C_{integer} ensures that:

$$C_{\text{integer}}(\mathbf{y}) = \mathbb{I}[y_i \in \mathbb{Z} \quad \forall y_i \in \mathbf{y}].$$

Example 2.2.3. Consider a system that assigns tasks to workers, where the number of tasks assigned to each worker must be an integer. Let $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ represent the values of the tasks assigned to each worker. The integer constraint ensures that:

$$C_{\text{integer}}(\mathbf{y}) = \mathbb{I}[y_i \in \mathbb{Z} \quad \forall y_i \in \mathbf{y}].$$

Logical Constraints: Logical constraints encode relationships between the values of different random variables, typically in the form of implications, conjunctions, or disjunctions. These constraints are crucial in neural-symbolic systems for capturing logical relationships between variable values, which may represent domain-specific rules, eligibility criteria, or decision-making logic.

Formally, a logical constraint C_{logic} is a Boolean function that evaluates whether a specific logical relationship holds between these values. For example, an implication between two conditions can be written as:

$$C_{\text{logic}}(\mathbf{y}) = \mathbb{I}[(\phi(\mathbf{y}) \rightarrow \psi(\mathbf{y}))],$$

Example 2.2.4. Consider a job application system where applicants must have at least five years of experience and a PhD to be eligible for a job. Let y_1 represent the number of years of experience, y_2 represent the level of education, and y_3 represent whether the applicant is eligible. The logical implication constraint can be written as:

$$C_{\text{logic}}(y_1, y_2, y_3) = \mathbb{I}[(y_1 \geq 5) \wedge (y_2 = \text{PhD}) \rightarrow (y_3 = 1)].$$

2.2.3 Soft Constraints

A soft constraint C_s defines a condition over the values of target variables $\mathbf{y} = \{y_1, y_2, \dots, y_m\}$ and, optionally, observed variables $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ or latent variables

$\mathbf{z} = \{z_1, z_2, \dots, z_p\}$.² Unlike hard constraints, soft constraints represent preferences or flexible conditions that a system aims to satisfy but which do not need to be strictly enforced. Formally, a soft constraint C_s is defined as a function that measures the degree of dissatisfaction or violation that returns a value in $\mathbb{R}_{\geq 0}$:

$$C_s(\mathbf{y}) \geq 0,$$

with $C_s(\mathbf{y}) = 0$ when the constraint is fully satisfied.

Example 2.2.5. *Consider the task of assigning employees shifts at a company, where each employee provides a ranked list of preferred shifts. The goal is to assign employees to shifts in a way that minimizes dissatisfaction with their preferences.*

Let x_i represent the employee’s ranked list of preferred shifts, and y_i represent the shift they are assigned. Define a soft constraint that assigns a penalty based on how far the assigned shift y_i is from the employee’s top preference in x_i . For instance, taking the difference in rank:

$$C_s(x_i, y_i) = |\text{Rank}(x_i) - \text{Rank}(y_i)|$$

where $C_s(x_i, y_i) = 0$ if the employee is assigned their top preference.

For instance, if an employee ranks their shifts as $\{\text{Shift}_1, \text{Shift}_3, \text{Shift}_2\}$, with Shift_1 being the most preferred, and they are assigned to Shift_3 , the soft constraint would evaluate to $C_s(\text{Shift}_1, \text{Shift}_3) = 1$.

2.2.4 Constraint Optimization

In many neural-symbolic systems, the goal is to find an optimal assignment of target variables \mathbf{y} that satisfies hard constraints while optimizing an objective function and handling soft constraints. This can be expressed as:

$$\arg \min_{\mathbf{y}} \mathcal{L}(\mathbf{y}) + \lambda \cdot C_s(\mathbf{y}),$$

²All definitions in this section can be straightforwardly extended to include either observed or latent variables.

subject to:

$$C_h(\mathbf{y}) = 1.$$

Here, $\mathcal{L}(\mathbf{y})$ is the objective function, $C_h(\mathbf{y})$ are the hard constraints, $C_s(\mathbf{y})$ are the soft constraints, and λ is a weighting value that controls the penalty for violating soft constraints.

Example 2.2.6. *Consider a resource allocation problem where a company must allocate resources to projects. Let \mathbf{x} represent the available resources and \mathbf{y} represent the amounts allocated to each project. The goal is to allocate resources optimally while satisfying both hard constraints, such as ensuring that no project exceeds its maximum resource limit, and soft constraints, such as minimizing the deviation from a preferred allocation.*

Hard constraint: Each project cannot exceed a maximum resource limit. Let x_i be the available resources for project i and y_i be the resources allocated to project i . The hard constraint is:

$$C_h(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } y_i \leq x_i \text{ for all } i, \\ 0 & \text{otherwise.} \end{cases}$$

Soft constraint: The company prefers a specific allocation, say $y_i = 0.5x_i$, but deviations are allowed at a penalty. The soft constraint is defined as:

$$C_s(\mathbf{x}, \mathbf{y}) = \sum_i |y_i - 0.5x_i|.$$

The objective function might be to maximize the total output of the projects:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = - \sum_i f(y_i),$$

where $f(y_i)$ is the expected output of project i as a function of the resources allocated.

Thus, the system seeks to minimize the combined objective:

$$\arg \min_{\mathbf{y}} \left(- \sum_i f(y_i) + \lambda \cdot \sum_i |y_i - 0.5x_i| \right),$$

subject to $y_i \leq x_i$ for all i .

2.2.5 Solving Constrained Optimization Problems

Constrained optimization problems have been widely studied in optimization theory. The approach to solving these problems depends on the nature of the objective function and the types of constraints involved.

2.2.5.1 Linear Programming

If both the objective function and the hard constraints are linear, and some hard constraints are inequalities, the problem is a *linear programming* problem. These problems can be solved using methods such as the simplex algorithm or interior-point methods. Linear programming problems are often solvable in polynomial time, making them efficient for large-scale optimization tasks.

Formally, a linear programming problem can be written as:

$$\arg \min_{\mathbf{y}} \mathbf{c}^\top \mathbf{y},$$

subject to $\mathbf{A}\mathbf{y} \leq \mathbf{b}$, where \mathbf{c} is a vector of coefficients and $\mathbf{A}\mathbf{y} \leq \mathbf{b}$ are the linear hard constraints.

2.2.5.2 Quadratic Programming

When the objective function is quadratic and the constraints are linear, the problem is called *quadratic programming*. Quadratic programming is a special case of nonlinear programming that can still be solved efficiently using methods such as the ellipsoid algorithm if the quadratic function is convex. However, if the objective function is non-convex, the problem may become NP-hard.

Quadratic programming problems are written as:

$$\arg \min_{\mathbf{y}} \frac{1}{2} \mathbf{y}^\top \mathbf{Q} \mathbf{y} + \mathbf{c}^\top \mathbf{y},$$

subject to $\mathbf{A}\mathbf{y} \leq \mathbf{b}$, where \mathbf{Q} is a symmetric matrix defining the quadratic terms and $\mathbf{A}\mathbf{y} \leq \mathbf{b}$ are the linear hard constraints.

2.3 NeSy Architectures

In this section, I propose four foundational architectural axioms that delineate how neural and symbolic components interact. The section begins with the *symbolic as neural structure* axiom (Section 2.3.1), which is a more neural-centric approach. Here, the structure of the neural network is derived directly from symbolic knowledge, embedding symbolic relationships and constraints as part of the network architecture. Next, I introduce the *sampling neural for symbolic* axiom (Section 2.3.2), which adopts a more symbolic perspective. This method extracts information from neural systems and integrates it into a non-differentiable symbolic approach. Finally, the focus shifts to architecture axioms with a holistic desire to maintain differentiability in that of *neural as symbolic parameter* (Section 2.3.3) and *neural as symbolic variable* (Section 2.3.4).

2.3.1 Symbolic as Neural Structure

The *symbolic as neural structure* architectural approach leverages symbolic knowledge to define the architecture of a neural network. In this paradigm, the neural network’s topology—including its input, output, hidden units, and the connections between them—is directly derived from the relationships and constraints inherent in a symbolic system. While the symbolic framework dictates the network structure, its parameters, representing the symbolic model’s relationships, are learned through typical neural data-driven training.

Neural-Symbolic Logic Example: A prominent application of the *symbolic as neural structure* approach involves translating a knowledge base containing Horn clause logical rules into a neural network, as exemplified in Knowledge-Based Artificial Neural Networks (KBANNs) [125]. The symbolic system provides the structural blueprint for the network, which is subsequently trained to approximate the reasoning encoded within the rules.

Architecture: Formally, let \mathbf{y} denote the target random variables, \mathbf{x}_{sy} the observed symbolic random variables, and \mathbf{z}_{sy} the latent symbolic random variables. A set of

grounded propositional logic constraints, $\mathbf{C}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{z}_{sy})$, is defined such that:

$$\mathbf{C}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{z}_{sy}) = \begin{cases} 1, & \text{if the constraints are satisfied,} \\ 0, & \text{if the constraints are violated.} \end{cases}$$

Based on these constraints, a neural model g_{nn}^{sy} is defined with a network topology that directly corresponds to $\mathbf{C}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{z}_{sy})$ and the associated random variables:

$$g_{nn}^{sy} : \mathbf{x}_{sy} \rightarrow \mathbf{y},$$

where the input nodes represent observed symbolic random variables \mathbf{x}_{sy} , the hidden nodes correspond to latent symbolic random variables \mathbf{z}_{sy} , and the output nodes map to the target random variables \mathbf{y} . Connections and weights are determined by the interactions encoded in the constraints $\mathbf{C}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{z}_{sy})$. A straightforward method to construct such a model is to exploit the directed nature of Horn clauses. These clauses provide a naturally directed path, where nodes corresponding to the body of a rule are connected to the nodes in its head. This structure defines a set of neural weights \mathbf{w}_{nn} assigned to each connection.

Example 2.3.1. *Consider the following Horn clause with observed random variables $\mathbf{x}_{sy} = \{x_A, x_B\}$ and target random variable y_C :*

$$x_A \wedge x_B \implies y_C.$$

The resulting neural network g_{nn}^{sy} has input units corresponding to the observed random variables x_A and x_B and an output unit corresponding to y_C . Connections within g_{nn}^{sy} form a dense layer from the inputs to the output, with neural weights $\mathbf{w}_{nn} = \{w_{AC}, w_{BC}\}$.

The implementation of the *symbolic as neural structure* approach varies significantly with the complexity and dependencies of the underlying knowledge base. For example, Knowledge-Based Artificial Neural Networks can extend the basic architecture by incorporating additional nodes and connections. It is also worth noting that inference and learning in such networks typically follow standard neural network methodologies, employing gradient-based optimization and backpropagation. These processes align with conventional neural training paradigms and are not explored in detail here.

2.3.2 Sampling Neural for Symbolic

In the *sampling neural for symbolic* architectural approach, there is a loose coupling between the neural network and symbolic reasoning. The neural network generates predictions, which are then subjected to a sampling process to produce symbolic elements, such as random variables, symbolic parameters, or constraints. This sampling process bridges the neural outputs and the symbolic reasoning system, enabling the symbolic system to operate independently of the neural network while leveraging its predictions as inputs or initial conditions. However, because the symbolic system is independent and non-differentiable, more complex training mechanisms for joint neural and symbolic parameter training, such as sampling-based learning, are required.

Neural-Symbolic Logic Example: A representative implementation of the *sampling neural for symbolic* approach is found in NeSy classical logic systems, where neural models define a probability distribution that samples boolean values for random variables in a classical logic framework [103]. Classical logic frameworks, such as propositional or first-order logic, impose hard constraints that enforce strict binary conditions on the system’s outputs, ensuring logical consistency. Note that the sampling process in this approach is not exclusive to random variables; for instance, it may be used to estimate symbolic parameters or even define the structure of the symbolic reasoning framework itself.

Architecture: Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} be the observed symbolic random variables, \mathbf{x}_{nn} be the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} be the symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model with parameters \mathbf{w}_{nn} and input \mathbf{x}_{nn} . Define a set of random variables \mathbf{q} , which are sampled from the neural network output $g_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$:

$$\mathbf{q} = \mathcal{P}(g_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})),$$

where \mathcal{P} is the sampling process applied to the neural network’s outputs. These sampled

random variables are then incorporated into the symbolic constraints:

$$C(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q}) = \begin{cases} 1, & \text{if the constraint is satisfied,} \\ 0, & \text{if the constraint is violated.} \end{cases}$$

Inference Scenario: A typical inference scenario for this NeSy logic setting is to solve a *Maximum Satisfiability (MAX-SAT)* problem [53]. The objective is to find an assignment \mathbf{y}^* for the target variables \mathbf{y} that maximizes the number of satisfied symbolic constraints $C(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})$:

$$\mathbf{y}^* = \arg \max_{\mathbf{y} \in \mathcal{Y}} \sum_{C_i \in \mathbf{C}} \mathbb{I}[C_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})],$$

To solve this MAX-SAT problem, methods such as *random walks* [101] can be used.

Learning Scenario: A typical learning scenario for this NeSy logic setting is challenging as the binary nature of C makes it *non-differentiable*:

$$\nabla_{\mathbf{w}_{nn}} C(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q}) \text{ is undefined for almost all } \mathbf{w}_{nn}.$$

A *Monte Carlo optimization* method, such as the *REINFORCE algorithm*, can be used [124]. Instead of requiring the loss function to be differentiable, these methods optimize the expected value of a reward function, which, in this case, reflects the satisfaction of the logical constraints.

Define a reward function $R(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})$, where the reward is positive if the logical constraints are satisfied and penalize violations:

$$R(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q}) = C(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q}) - \lambda \mathcal{L}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q}),$$

where \mathcal{L} is a task-specific loss function (such as cross-entropy or mean squared error), and λ is a weighting factor that balances the satisfaction of the logical constraints and the main prediction task. The objective becomes maximizing the expected reward:

$$\mathbb{E}[R(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})].$$

Monte Carlo sampling allows us to approximate the gradient of this expected reward with respect to the network parameters \mathbf{w}_{nn} :

$$\nabla_{\mathbf{w}_{nn}} \mathbb{E}[R(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})] \approx \frac{1}{N} \sum_{i=1}^N R(\mathbf{y}, \mathbf{x}_i, \mathbf{q}_i) \nabla_{\mathbf{w}_{nn}} \log p_{\mathbf{w}_{nn}}(\mathbf{q}_i),$$

where N is the number of samples and $p_{\mathbf{w}_{nn}}(\mathbf{q})$ is the probability of the network producing output \mathbf{q} .

Example 2.3.2. Consider the MNIST-Addition example discussed above (Example 2.1.1), where the task is to predict the sum of two digits presented as MNIST images. In that example, the model performs an $\arg \max$ and is only used for NeSy inference (i.e., predicting the sum). If the goal is instead to train the parameters of the digit classifier neural model, then instead of using the non-differentiable $\arg \max$ to select the most likely digits, we can sample from these probability distributions to obtain candidate digits for addition.

Concretely, given two MNIST images ($\mathbf{3}$, $\mathbf{5}$) with the target sum 8. The images are passed through the neural network g_{nn} , yielding probability distributions over the digits:

$$\begin{aligned} g_{nn}(\mathbf{3}) = \mathbf{q}_1 &= \{P(0)_1 = 0.0, P(1)_1 = 0.0, P(2)_1 = 0.0, P(3)_1 = 0.9, P(4)_1 = 0.0, \\ &\quad P(5)_1 = 0.0, P(6)_1 = 0.0, P(7)_1 = 0.0, P(8)_1 = 0.1, P(9)_1 = 0.0\}, \\ g_{nn}(\mathbf{5}) = \mathbf{q}_2 &= \{P(0)_2 = 0.0, P(1)_2 = 0.0, P(2)_2 = 0.0, P(3)_2 = 0.0, P(4)_2 = 0.0, \\ &\quad P(5)_2 = 0.8, P(6)_2 = 0.0, P(7)_2 = 0.0, P(8)_2 = 0.0, P(9)_2 = 0.2\}. \end{aligned}$$

Let $\mathbf{q}_1 \sim g_{nn}(\mathbf{3})$ and $\mathbf{q}_2 \sim g_{nn}(\mathbf{5})$ represent the sampled digits. The sum of these sampled digits, $\mathbf{y}_{sum} = \mathbf{q}_1 + \mathbf{q}_2$, is compared against the target value 8 to compute a reward and the gradient of the expected reward can be approximated:

$$R(\mathbf{q}_1, \mathbf{q}_2, \mathbf{y}_{sum}) = \begin{cases} 1, & \text{if } \mathbf{y}_{sum} = 8, \\ 0, & \text{otherwise.} \end{cases}$$

While Monte Carlo methods like REINFORCE provide a viable solution to the non-differentiability issue, they bring their own set of challenges. The inherent stochasticity of sampling often leads to *high variance*, which can slow convergence and introduce

instability during training. While this is a viable solution, much of the literature addresses this challenge by introducing probabilistic logic interpretations (Section 2.3.3) and/or by softening with differentiable fuzzy logics (Section 2.3.4).

2.3.3 Neural as Symbolic Parameter

In the *neural as symbolic parameter* architectural approach, the neural network predicts parameters—such as probabilities or weights—that influence a differentiable symbolic reasoning framework. Unlike architectures where neural networks directly output variables for symbolic reasoning, this approach relies on the network to guide the symbolic model’s behavior through the parameters. This design preserves a level of independence between the neural and symbolic components while enabling a differentiable structure to support training.

Neural-Symbolic Logic Example: A representative implementation of the *neural as symbolic parameter* approach is found in NeSy probabilistic logic systems, where neural models define the probabilities over facts [81, 138, 141]. Probabilistic logic extends classical logic by introducing uncertainty by assigning probabilities to facts or rules. For instance, a probabilistic fact might state that “Alice smokes” with a probability of 0.7, denoted as $0.7 :: \text{Smokes}(\text{“Alice”})$. In NeSy probabilistic systems, these probabilities are defined by a neural network.

Architecture: Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} be the observed symbolic random variables, \mathbf{x}_{nn} be the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} be the symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model with parameters \mathbf{w}_{nn} and input \mathbf{x}_{nn} . The neural network output \mathbf{q} is used to define a distribution $p_{\mathbf{w}_{nn}}(w|\mathbf{q})$ over worlds $w \in \{0, 1\}^n$. The symbolic logic constraints, $C(w)$, are Boolean functions that determine whether a world is considered valid, i.e., $C(w) = 1$ if the world satisfies the constraint and $C(w) = 0$ otherwise.

To simplify learning and inference, many approaches adopt the *conditional independence assumption*, which assumes that the variables w_i are conditionally independent given the neural input \mathbf{x}_{nn} . Under this assumption, the joint probability $p_{\mathbf{w}_{nn}}(w|\mathbf{q})$ can be

factorized as:

$$p_{\mathbf{w}_{nn}}(w|\mathbf{q}) = \prod_{i=1}^n p_{\mathbf{w}_{nn}}(w_i|\mathbf{q}).$$

Inference Scenario: A typical inference scenario for this NeSy probabilistic logic setting involves computing the probability that a given constraint C is satisfied, often referred to as the *weighted model count*:



$$p_{\mathbf{w}_{nn}}(C = 1|\mathbf{x}_{nn}) = \sum_{w \in \{0,1\}^n} p_{\mathbf{w}_{nn}}(w|\mathbf{q}) C(w),$$

The weighted model count sums the probability over all possible worlds $w \in \{0,1\}^n$ where the constraint holds.

Learning Scenario: A typical learning scenario for this NeSy probabilistic logic setting involves minimize the *negative logarithm* of the weighted model count:

$$\mathcal{L}(\mathbf{w}_{nn}; \mathbf{x}_{nn}) = -\log p_{\mathbf{w}_{nn}}(C = 1|\mathbf{x}_{nn}).$$

Example 2.3.3. Consider the MNIST-Addition example discussed above (Example 2.1.1), where the task is to predict the sum of two digits presented as MNIST images. In that example, the model performs an $\arg \max$ to choose values deterministically for NeSy inference (i.e., predicting the sum). However, if the goal is to train the parameters of the digit classifier neural model using probabilistic logics, we can use weighted model counting to compute the probability of the correct sum instead of relying on the non-differentiable $\arg \max$.

For instance, given two MNIST images (, ) , with the correct sum as 8. This means that the constraint $C(w)$ is true only when the sum of the digits in world w equals 8. The images are passed through the neural network g_{nn} , which outputs probability distributions over the possible digits for each image:

$$\begin{aligned} g_{nn}(\text{3}) &= \mathbf{q}_1 = \{P(0)_1 = 0.0, P(1)_1 = 0.0, P(2)_1 = 0.0, P(3)_1 = 0.9, P(4)_1 = 0.0, \\ &\quad P(5)_1 = 0.0, P(6)_1 = 0.0, P(7)_1 = 0.0, P(8)_1 = 0.1, P(9)_1 = 0.0\}, \\ g_{nn}(\text{5}) &= \mathbf{q}_2 = \{P(0)_2 = 0.0, P(1)_2 = 0.0, P(2)_2 = 0.0, P(3)_2 = 0.0, P(4)_2 = 0.0, \\ &\quad P(5)_2 = 0.8, P(6)_2 = 0.0, P(7)_2 = 0.0, P(8)_2 = 0.0, P(9)_2 = 0.2\}. \end{aligned}$$

Assuming conditional independence between the digits from each image, the weighted model count sums the probabilities of pairs of values that satisfy the constraint $C(w)$ (i.e., sum to 8):

$$\begin{aligned} p_{\mathbf{w}_{nn}}(C(w) = 1|\mathbf{q}_1, \mathbf{q}_2) = & P(0)_1 \cdot P(8)_2 + P(1)_1 \cdot P(7)_2 + P(2)_1 \cdot P(6)_2 + P(3)_1 \cdot P(5)_2 + \\ & P(4)_1 \cdot P(4)_2 + P(5)_1 \cdot P(3)_2 + P(6)_1 \cdot P(2)_2 + P(7)_1 \cdot P(1)_2 + \\ & P(8)_1 \cdot P(0)_2. \end{aligned}$$

Substituting the probabilities from the neural network outputs:

$$\begin{aligned} p_{\mathbf{w}_{nn}}(C(w) = 1|\mathbf{q}_1, \mathbf{q}_2) = & (0.0 \cdot 0.0) + (0.0 \cdot 0.0) + (0.0 \cdot 0.0) + (0.9 \cdot 0.8) + \\ & (0.0 \cdot 0.0) + (0.0 \cdot 0.0) + (0.0 \cdot 0.0) + (0.0 \cdot 0.0) + \\ & (0.1 \cdot 0.0) \\ = & 0.9 \cdot 0.8 = 0.72. \end{aligned}$$

The loss is then computed as the negative log-likelihood of the constraint being satisfied:

$$\begin{aligned} \mathcal{L}(\mathbf{w}_{nn}; \mathbf{q}_1, \mathbf{q}_2) = & -\log p_{\mathbf{w}_{nn}}(C(w) = 1|\mathbf{q}_1, \mathbf{q}_2) \\ = & -\log 0.72 \approx 0.33. \end{aligned}$$

By minimizing this loss, the neural network learns to increase the probability of generating outputs that satisfy the probabilistic logic constraint (i.e., the sum of the digits equals 8).

While probabilistic logics offer flexibility in handling uncertainty and soft constraints, they also introduce computational challenges. Summing over all possible worlds for probabilistic inference can be computationally expensive, especially as the number of variables, n , grows large. Exact inference in these settings is generally intractable. However, knowledge compilation methods transform the problem into an equivalent form that can be computed in constant time (once compiled). This transformation, which can be performed in polynomial time, allows for efficient inference even in complex models. Both weighted

model counting and the knowledge compilation are differentiable, enabling gradient-based optimization and backpropagation. Furthermore, the conditional independence assumption helps reduce the number of parameters and simplifies the inference process. Nevertheless, this assumption may oversimplify complex dependencies between variables, potentially limiting the expressiveness of the model in capturing rich relationships.

2.3.4 Neural as Symbolic Variable

In the *neural as symbolic variable* architectural approach, the outputs of a neural network are treated as random variables within a differentiable symbolic reasoning model. Unlike the previous architecture, this approach allows the neural model to have a direct influence on the satisfaction of symbolic constraints. This design achieves a tighter integration between the neural and symbolic components and can be viewed as an extension of the neural architecture with a symbolic optimization layer.

Neural-Symbolic Logic Example: A notable implementation of the *neural as symbolic variable* approach is seen in NeSy fuzzy logic systems, where neural models define random variables in a differentiable fuzzy logic framework [14, 103]. Fuzzy logic extends classical logic by allowing truth values to range continuously between 0 and 1, enabling reasoning over partial truths. For example, a fuzzy rule like $A \wedge \neg B$ can be approximated by the differentiable function $A \cdot (1 - B)$.

Note 2.3.1. *NeSy fuzzy logic systems allow for numerous interpretations of logical operations and their combinations. Among the various t-norms, three commonly studied within NeSy are Gödel, Product, and Łukasiewicz Table 2.1. These t-norms adhere to fundamental properties such as commutativity, associativity, monotonicity, and unit [52]. The choice of*

Logic	NEGATION ($\neg x$)	AND ($x \wedge y$)	OR ($x \vee y$)
Minimum/Gödel	$1 - x$	$\min(x, y)$	$\max(x, y)$
Product	$1 - x$	$x \cdot y$	$x + y - x \cdot y$
Łukasiewicz	$1 - x$	$\max(0, x + y - 1)$	$\min(1, x + y)$

Table 2.1: Fuzzy/Soft Logic Constraints for NEGATION, AND, and OR.

fuzzy logic operators, particularly for conjunction (AND), is critical for learning and optimization tasks. Studies by van Krieken et al. (2023) and Evans and Grefenstette (2018) have extensively evaluated the effectiveness of different fuzzy logic operators in diverse applications. The following observations are noteworthy:

- *Gödel t-norm:* For $x > y$, the conjunction $\min(x, y)$ passes no gradient to x , as the operation depends only on the smaller value. This can hinder gradient propagation during optimization.
- *Lukasiewicz t-norm:* When $x + y < 1$, the conjunction $\max(0, x + y - 1)$ provides no gradient information to either x or y , limiting the ability to update model parameters in cases with low input values.
- *Product t-norm:* The conjunction $x \cdot y$ distributes gradients proportionally between x and y , ensuring that both inputs contribute to the optimization process. This property makes the Product t-norm more favorable for gradient-based learning.

Furthermore, NeSy fuzzy logics allow for different aggregation operations:

- *Weighted Sum:* A linear combination of dissatisfaction, weighted by their relative importance:

$$\text{Weighted Sum} = \sum_{i=1}^n w_i s_i,$$

where w_i is the weight of the dissatisfaction s_i .

- *Generalized Mean:* Extends the arithmetic mean by allowing control over how strictly lower values penalize the aggregated score:

$$\text{Generalized Mean} = \left(\frac{1}{n} \sum_{i=1}^n s_i^p \right)^{\frac{1}{p}},$$

where p determines the penalty imposed by low values, with smaller p penalizing low values more aggressively.

Architecture: Formally, let \mathbf{y} denote the target random variables, \mathbf{x}_{sy} the observed symbolic random variables, \mathbf{x}_{nn} the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} the

symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model parameterized by \mathbf{w}_{nn} with input \mathbf{x}_{nn} . The neural network output, \mathbf{q} , defines a subset of the random variables:

$$\mathbf{q} = g_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}),$$

which are then integrated into symbolic differentiable soft logic constraints $C(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{q})$.

Inference Scenario: A typical learning scenario for this NeSy fuzzy logic setting involves finding the Maximum A Posteriori (MAP) estimate of the target variables \mathbf{y} . Assuming a weighted sum aggregation for the soft logic constraints, the inference problem can be formulated as follows:

$$\mathbf{y}^* = \arg \max_{\mathbf{y}} \sum_{i=1}^n w_{sy}^i C_i(\mathbf{y}, \mathbf{x}, \mathbf{q}),$$



where w_{sy}^i are the symbolic weights associated with each constraints C_i .

Learning Scenario: A typical learning scenario for this NeSy fuzzy logic setting has the neural and symbolic parameters optimized to minimize the dissatisfaction of the fuzzy constraints. The learning optimization problem is expressed as follows:

$$\mathbf{w}_{nn}^*, \mathbf{w}_{sy}^* = \arg \min_{\mathbf{w}_{nn}, \mathbf{w}_{sy}} \mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{sy}; \mathbf{x}_{nn}, \mathbf{x}_{sy}, \mathbf{y}),$$

where $\mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{sy}; \mathbf{x}_{nn}, \mathbf{x}_{sy}, \mathbf{y})$ is the loss function capturing the degree of constraint dissatisfaction.

Example 2.3.4. *Consider the MNIST-Addition task, where the goal is to predict the sum of two digits presented as MNIST images using Gödel fuzzy logic. In this setup, the neural model g_{nn} predicts the truth values of random variables representing the digits, and the Gödel fuzzy logic system enforces constraints regarding their sum.*

For instance, given two MNIST images,  and , the target sum is 8. A soft logic constraint representing this addition is:

$$q(x_1)_1 \wedge q(x_2)_2 \rightarrow \text{Sum}(x_1, x_2, 8),$$

where $q(x_1)_1$ and $q(x_2)_2$ represent the truth values of the variables corresponding to the digits output by the neural network and $\text{Sum}(x_1, x_2, 8)$ is valid if the sum of the two values

sum to eight. Suppose the neural model g_{nn} provides the following outputs:

$$\begin{aligned} g_{nn}(\mathbf{3}) = \mathbf{q}_1 &= \{q(0)_1 = 0.0, q(1)_1 = 0.0, q(2)_1 = 0.0, q(3)_1 = 0.9, q(4)_1 = 0.0, \\ & q(5)_1 = 0.0, q(6)_1 = 0.0, q(7)_1 = 0.0, q(8)_1 = 0.1, q(9)_1 = 0.0\}, \\ g_{nn}(\mathbf{5}) = \mathbf{q}_2 &= \{q(0)_2 = 0.0, q(1)_2 = 0.0, q(2)_2 = 0.0, q(3)_2 = 0.0, q(4)_2 = 0.0, \\ & q(5)_2 = 0.8, q(6)_2 = 0.0, q(7)_2 = 0.0, q(8)_2 = 0.0, q(9)_2 = 0.2\}. \end{aligned}$$

Gödel fuzzy logic evaluates the satisfaction of the constraint by aggregating the truth values of all valid digit pairs whose sum equals 8. Using the fuzzy conjunction operator, the overall satisfaction value is computed as:

$$C(w) = \sum_{(i,j): i+j=8} \min(q(i)_1, q(j)_2),$$

where each pair (i, j) corresponds to digits that sum to 8. Substituting the neural outputs:

$$\begin{aligned} C(w) &= \min(q(0)_1, q(8)_2) + \min(q(1)_1, q(7)_2) + \min(q(2)_1, q(6)_2) + \\ & \min(q(3)_1, q(5)_2) + \min(q(4)_1, q(4)_2) + \min(q(5)_1, q(3)_2) + \\ & \min(q(6)_1, q(2)_2) + \min(q(7)_1, q(1)_2) + \min(q(8)_1, q(0)_2) \\ &= \min(0.0, 0.0) + \min(0.0, 0.0) + \min(0.0, 0.0) + \\ & \min(0.9, 0.8) + \min(0.0, 0.0) + \min(0.0, 0.0) + \\ & \min(0.0, 0.0) + \min(0.0, 0.0) + \min(0.1, 0.0) \\ &= 0.8. \end{aligned}$$

It is important to recognize that the previous example involves several key decisions, such as the choice of soft logic representation, the aggregation method, and the simplification of inference as a direct calculation without latent variables. These decisions underscore both the inherent complexity and the flexibility of fuzzy NeSy logic, illustrating why it is challenging to provide a universal, one-size-fits-all framework for such systems.

Part II

Universal Neural-Symbolic Language

Chapter 3

Unifying NeSy through Energy-Based Models

The previous chapter introduced a set of architectural principles that form the foundation of Neural-Symbolic (NeSy) AI. Building on this, this chapter formalizes these concepts by presenting a unifying language for NeSy systems. Specifically, I introduce *Neural-Symbolic Energy-Based Models* (NeSy-EBMs), a comprehensive mathematical framework that encapsulates the diverse facets of neural-symbolic methods and their applications. NeSy-EBMs formalize the interaction between neural and symbolic components as a structured composition of functions, providing a robust and flexible framework for analyzing, integrating, and extending these paradigms.

This chapter is organized as follows: first, I define Neural-Symbolic Energy-Based Models (Section 3.1); second, I formalize the modeling paradigms that serve as integration strategies for the architectural designs introduced in the previous chapter (Section 3.2); and finally, I provide a formalization of several mainstream NeSy approaches as specific instantiations of the NeSy-EBM framework (Section 3.3). The theory and notation introduced here will serve as a foundation throughout this thesis, guiding the analysis and discussion of hard and soft constraints within neural-symbolic systems.

3.1 Neural Symbolic Energy-Based Models as a Unifying Mathematical Framework for NeSy

As the name suggests, NeSy-EBMs build upon the foundations of Energy-Based Models (EBMs) [74]. In particular, NeSy-EBMs are a family of EBMs that integrate deep architectures with explicit encodings of symbolic relations via an energy function. EBMs are a class of models where an energy function is used to evaluate the compatibility between variables; states with lower energy correspond to higher compatibility. Specifically, in the context of NeSy-EBMs, a low-energy state indicates high compatibility, meaning that the variables align well with both domain-specific knowledge and common sense.

As diagrammed in Figure 3.1, a NeSy-EBM energy function composes a neural component with a symbolic component, represented by the functions \mathbf{g}_{nn} and \mathbf{g}_{sy} , respectively. The neural component is a deep model (or collection of deep models) parameterized by weights from a domain \mathcal{W}_{nn} , that takes a neural input from a domain \mathcal{X}_{nn} and outputs a real-valued vector of dimension d_{nn} . The symbolic component encodes domain knowledge and is parameterized by weights from a domain \mathcal{W}_{sy} . It maps inputs from a domain \mathcal{X}_{sy} , target (or output) variables from \mathcal{Y} , and neural outputs from $\text{Range}(\mathbf{g}_{nn})$ to a scalar value. In other words, the symbolic component measures the compatibility of targets, inputs, and neural outputs with domain knowledge. We have the following formal definition:

Definition 1. *A **NeSy-EBM energy function** is a mapping parameterized by neural and symbolic weights from domains \mathcal{W}_{nn} and \mathcal{W}_{sy} , respectively, that quantifies the compatibility of a target variable from a domain \mathcal{Y} and neural and symbolic inputs from the domains \mathcal{X}_{nn} and \mathcal{X}_{sy} , respectively, with a scalar value:*

$$E : \mathcal{Y} \times \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn} \rightarrow \mathbb{R}. \quad (3.1)$$

*A NeSy-EBM energy function is a composition of a **neural** and **symbolic component**. Neural weights parameterize the neural component, which outputs a real-valued vector of dimension d_{nn} :*

$$\mathbf{g}_{nn} : \mathcal{X}_{nn} \times \mathcal{W}_{nn} \rightarrow \mathbb{R}^{d_{nn}}. \quad (3.2)$$

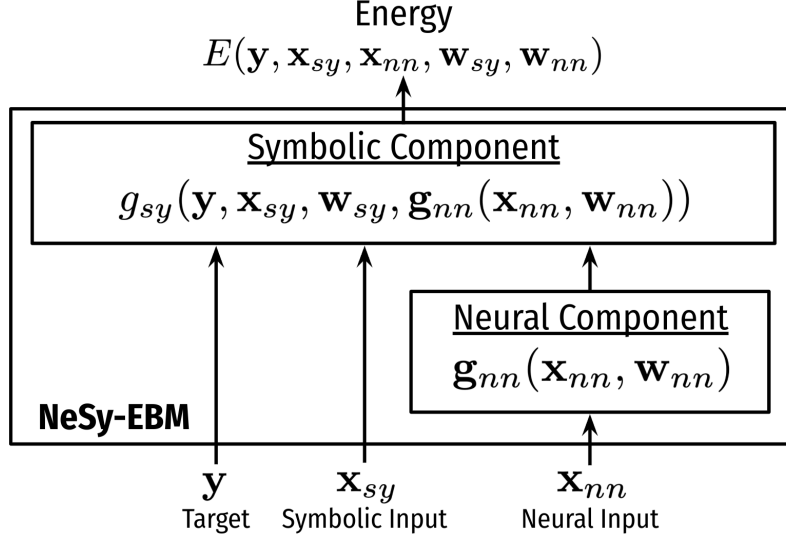


Figure 3.1: A neural-symbolic energy-based model.

The symbolic component maps the symbolic variables, symbolic parameters, and a real-valued vector of dimension d_{nn} to a scalar value:

$$g_{sy} : \mathcal{Y} \times \mathcal{X}_{sy} \times \mathcal{W}_{sy} \times \mathbb{R}^{d_{nn}} \rightarrow \mathbb{R}. \quad (3.3)$$

The NeSy-EBM energy function is

$$E : (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \mapsto g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \quad \square$$

What I have introduced thus far can be understood as the energy function that defines a NeSy approach, forming the core representation of the system. However, this does not yet define the reasoning or inference processes that operate within this framework. In the following chapter (Chapter 4), I will formally introduce the mechanisms for performing inference and learning in NeSy-EBMs, along with typical methods by which these systems execute reasoning and learning tasks. The remainder of this chapter is devoted to the structural construction of a NeSy-EBM, specifically addressing how its architectural axioms are represented. This includes detailing the interaction between neural and symbolic components, providing the foundation for understanding how these models function in practice, and preparing for the forthcoming discussions on inference and learning.

3.2 NeSy-EBM Modeling Paradigms

Using the NeSy-EBM framework, this subsection introduces a taxonomy of NeSy modeling paradigms determined by the nature of the neural-symbolic interface. The paradigms are characterized by the integration of the neural component within the symbolic component to define the prediction program in Equation 4.1. Conceptually, these categorizations represent standard NeSy architectural designs that describe the interaction between neural and symbolic components.

To formalize these paradigms, I introduce an additional layer of abstraction, termed *symbolic potentials* and denoted by ψ . Furthermore, a collection of symbolic potentials forms what is referred to as *symbolic potential sets*, denoted by Ψ . These symbolic potentials categorize the arguments of the symbolic component based on their roles in shaping the prediction program in Equation 4.1.

Definition 2. A *symbolic potential* ψ is a function of variables from a domain V_ψ and parameters from a domain $Params_\psi$, outputting a scalar value:

$$\psi : V_\psi \times Params_\psi \rightarrow \mathbb{R}. \quad (3.4)$$

A *symbolic potential set*, denoted by Ψ , is a set of potential functions indexed by \mathbf{J}_Ψ . \square

A *modeling paradigm* is a specification of the set of symbolic potentials and the domains of the potentials belonging to the set. This section describes three modeling paradigms in the following subsections: *deep symbolic variables (DSVar)*, *deep symbolic parameters (DSPar)*, and *deep symbolic potentials (DSPot)*.

3.2.1 Deep Symbolic Variables

The deep symbolic variables (DSVar) paradigm is a NeSy-EBM approach in which the neural component predicts the values of target or latent variables used in a symbolic potential.¹ This modeling paradigm defines NeSy-EBMs that can represent the *sampling neural for symbolic* architectural axiom (Section 2.3.2) and the *neural as symbolic variable*

¹This section focuses on deep symbolic variables in the context of target variables. Extending to latent variables is straightforward.

architectural axiom (Section 2.3.4). DSVar has a one-to-one mapping between the neural outputs and the target or latent variables. However, this mapping is not necessarily *onto*, meaning there may be target or latent variables that do not correspond to any neural output. Prominent NeSy systems exemplifying this paradigm include Logic Tensor Networks [14], Learning with Logical Constraints [57], Semantic-Based Regularization [46], and Deep Logic Models [88].

Definition 3. *In the **deep symbolic variables** (DSVar) modeling paradigm the symbolic potential set is a singleton $\Psi = \{\psi\}$ with a trivial index set $\mathbf{J}_\Psi = \{1\}$ such that $\Psi_1 = \psi$. Further, the neural prediction is treated as a variable by the symbolic potential; thus $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy} \times \mathbb{R}^{d_{nn}}$. Then, the symbolic parameters are the symbolic weights, $Params_\psi = \mathcal{W}_{sy}$. The neural component controls the NeSy-EBM prediction via this function:*

$$I_{\mathcal{Y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \begin{cases} 0 & \mathbf{y}_i = [\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]_i, \forall i \in \{1, \dots, d_{nn}\} \\ \infty & o.w. \end{cases}, \quad (3.5)$$

where \mathbf{y}_i and $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})_i$ denote the i 'th entry of the variable and neural output vectors, respectively. Then, the symbolic component expressed via the symbolic potential is:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \psi([\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})], \mathbf{w}_{sy}) + I_{\mathcal{Y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})),$$

where $[\cdot]$ denotes concatenation. □

The DSVar modeling paradigm typically yields the most straightforward prediction program compared to the other modeling paradigms. This is because the neural model fixes a subset of the decision or latent variables, making the prediction program smaller. This is achieved by adding the function (Equation 3.5) in the definition above to the symbolic potential so infinite energy is assigned to variable values that do not match the neural model's predictions. While this simplifies the prediction program and can be used to speed up inference and learning, this may result in a symbolic component having situations where it is impossible to resolve all constraint violations. Rather, DSVar models may rely on learning to train a neural component to adhere to constraints. The DSVar paradigm is demonstrated in the following example.

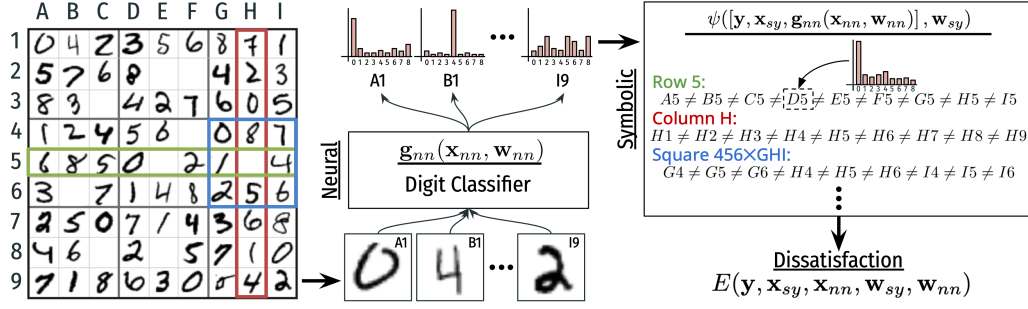


Figure 3.2: A deep symbolic variables model for solving a Sudoku board constructed from handwritten digits. The neural component classifies handwritten digits. Then, the symbolic component uses the digit classifications and the rules of Sudoku to fill in the empty cells.

Example 3.2.1. *Visual Sudoku [132] puzzle solving is the problem of recognizing handwritten digits in non-empty puzzle cells and reasoning with the rules of Sudoku (no repeated digits in any row, column, or box) to fill in empty cells. Figure 3.2 shows a partially complete Sudoku puzzle created with MNIST images [75] and a NeSy-EBM designed for visual Sudoku solving. The neural component is a digit classifier predicting the label of MNIST images, and the symbolic component quantifies rule violations.*

Formally, the target variables, \mathbf{y} , are the categorical labels of both the handwritten digits and the puzzle’s empty entries. The symbolic inputs, \mathbf{x}_{sy} , indicate whether two puzzle positions are in the same row, column, or box. The neural model, $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, is the categorical labels of the handwritten digits predicted by the neural component. Then, the symbolic parameters, \mathbf{w}_{sy} , are used to shape the single symbolic potential function, ψ , that quantifies the total amount of Sudoku rule violations.

The DSVar modeling paradigm is specifically designed to allow the neural component to directly influence the random variables within the symbolic model. While this paradigm allows for direct influence on the predictions of a symbolic model, its scope is strictly confined to random variables. In scenarios where the neural model must exert indirect influence on variables or interact with other elements of the symbolic model—such as entire symbolic potentials or parameters associated with individual constraints—a different

modeling paradigm becomes necessary. The following subsection introduces a paradigm that extends the neural component’s influence, enabling connections to other parameters or constants within the symbolic model, beyond just the random variables.

3.2.2 Deep Symbolic Parameters

The deep symbolic parameters (DSPar) modeling paradigm is a NeSy-EBM in which the neural component predicts the parameters of random variables or symbolic potentials. This modeling paradigm defines NeSy-EBMs that can represent the *sampling neural for symbolic* architectural axiom (Section 2.3.2) and the *neural as symbolic parameter* architectural axiom (Section 2.3.3). Prominent NeSy frameworks supporting this technique include DeepProbLog [81], Semantic Probabilistic Layers [7], and Semantic Loss [138].

Definition 4. *In the **deep symbolic parameters** (DSPar) modeling paradigm, the symbolic potential set is a singleton $\Psi = \{\psi\}$ with a trivial index set $\mathbf{J}_\Psi = \{1\}$ such that $\Psi_1 = \psi$. Further, the neural prediction is treated as a parameter by the symbolic potential, thus $\text{Params}_\psi = \mathcal{W}_{sy} \times \mathbb{R}^{d_{nn}}$. Then the symbolic variables are the targets and the symbolic inputs: $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$. The symbolic component expressed via the single symbolic potential is:*

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \psi([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]). \quad \square$$

This paradigm is demonstrated in the following example.

Example 3.2.2. *Citation network node classification is the task of predicting the topic of papers in a network where nodes are papers and edges are citations. A model may use the network structure, content, and topic labels in a paper’s neighborhood for prediction. Figure 3.3 shows a citation network and a NeSy-EBM designed for node classification. The symbolic component is a collection of weighted arithmetic constraints. One constraint represents the heuristic that two papers connected by a citation in the network have the same topic, and another uses a neural network to predict a topic for each paper using only its content. If a constraint is violated by a random variable, then a cost proportional to the weight of the violated constraint will be added to the energy function. The neural component*

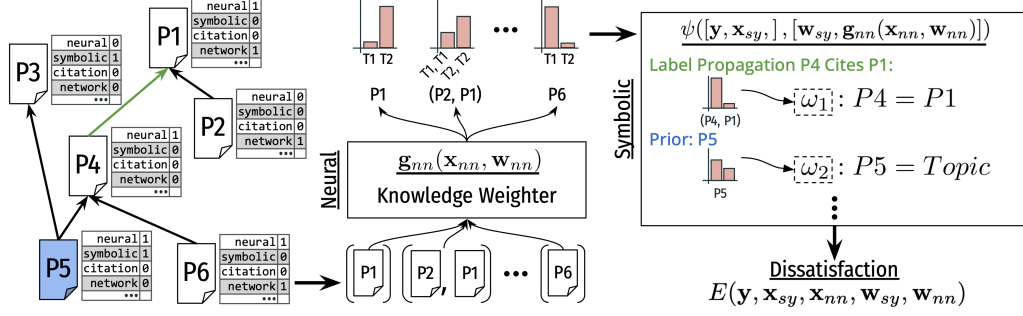


Figure 3.3: A deep symbolic parameters model for citation network node classification. The symbolic component is a mixture of experts model that combines weighted arithmetic constraints. The neural component uses paper content to weigh the importance of satisfying an arithmetic constraint.

predicts the weights of each constraint given the paper content. In other words, the NeSy-EBM is a mixture of experts. Each arithmetic constraint in the symbolic component is an expert weighted by the neural component, and experts are combined to formulate a mathematical program to produce a single output.

Formally, the targets, \mathbf{y} , are the paper topics, and the symbolic inputs, \mathbf{x}_{sy} , are citation links. The neural model, $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, predicts the weights of every constraint.

The DSPar modeling paradigm is widely applicable. For instance, the DSPar modeling paradigm is applied for constraint satisfaction, fine-tuning, few-shot, and semi-supervised settings in our empirical analysis. Note, however, that DSVar and DSPar models have only a single fixed symbolic potential. This property makes these paradigms well-suited for dedicated tasks but less applicable to open-ended settings, where the relevant domain knowledge depends on context. To address this challenge, the following modeling paradigm leverages generative modeling to perform in open-ended tasks.

3.2.3 Deep Symbolic Potentials

The deep-symbolic potentials (DSPot) paradigm is a NeSy-EBM approach in which a deep model predicts what symbolic structure should be used. This modeling

paradigm defines NeSy-EBMs that can represent the *sampling neural for symbolic* architectural axiom (Section 2.3.2). At a high level, the neural component is a generative model that samples symbolic potentials from a set to define the symbolic component. The Logic-LM pipeline proposed by Pan et al. (2023) is an excellent example of this modeling paradigm.

Definition 5. In the *deep symbolic potentials* modeling paradigm, the symbolic potential set Ψ is the set of all potential functions that can be created by a NeSy framework. Ψ is indexed by the output of the neural component, i.e., $\mathbf{J}_\Psi = \text{Range}(\mathbf{g}_{nn})$ and $\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}$ is the potential function indexed by the neural prediction. The variable and parameter domains of the sampled symbolic potential are $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$, and $\text{Params}_\psi = \mathcal{W}_{sy}$, respectively. The symbolic component expressed via the symbolic potential is:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy}). \quad \square$$

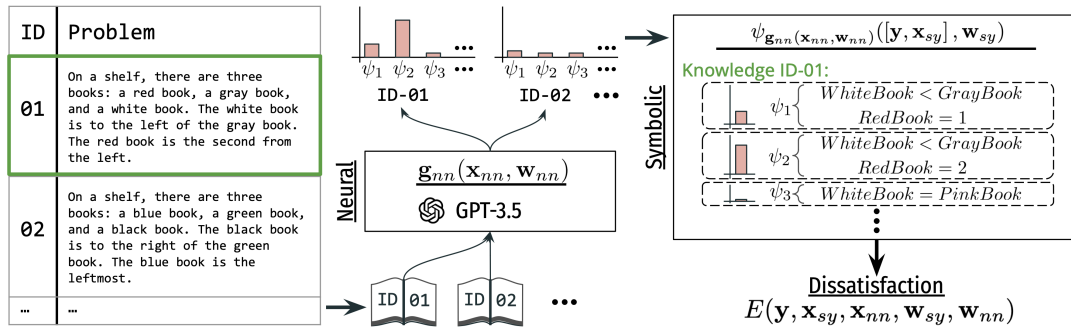


Figure 3.4: A deep symbolic potential model for answering questions about a set of objects' order described in natural language. The neural component is an LLM that generates syntax to create a symbolic potential. The symbolic potential is used to perform deductive reasoning and answer the question. See Example 3.2.3 for details.

This paradigm is demonstrated in the following example.

Example 3.2.3. *Question answering is the problem of giving a response to a question posed in natural language. Figure 3.4 shows a set of word problems asking for the order of a set of objects given information expressed in natural language and a NeSy-EBM designed*

for question answering. The neural component is a large language model (LLM) that is prompted with a word problem and tasked with generating a program within the syntax of a symbolic framework. The symbolic framework uses the generated program to instantiate a symbolic component used to perform deductive reasoning.

Formally, the target variables, \mathbf{y} , represent object positions, and there is no symbolic input, \mathbf{x}_{sy} , in this example. The neural input, \mathbf{x}_{nn} , is a natural language prompt that includes the word problem. The neural model, $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, is an LLM that generates syntax for a declarative symbolic modeling framework that creates the symbolic potential. For instance, the symbolic potential generated by the neural model $\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy})$ could be the total amount of violation of arithmetic constraints representing ordering. Finally, the symbolic parameters, \mathbf{w}_{sy} , shape the symbolic potential function.

DSPot NeSy-EBMs are the only applicable paradigms for truly open-ended tasks. Moreover, DSPot enhances generative models, such as LLMs, with consistent symbolic reasoning capabilities. DSPot’s limitation is that the neural component must learn to sample from a large potential set. For instance, in the example, an LLM must reliably generate syntax to define a symbolic potential for solving the word problem. LLMs require a substantial amount of computational resources to train and then fine-tune for a specific NeSy framework. Furthermore, the inference time is dependent on the sampled symbolic potential. If the neural component samples a complex symbolic potential, inference may be slow.

3.3 Expressing NeSy Approaches via NeSy-EBMs

As introduced in the previous section, NeSy-EBMs provide a general theoretical framework for describing neural-symbolic methods. This section highlights three prominent neural-symbolic approaches, exploring how they integrate neural and symbolic learning and reasoning through the lens of NeSy-EBMs. At a high level, these approaches differ in how they incorporate symbolic knowledge, constraints, and their overall modeling approach:

- **Semantic Loss (SL)** [138]: A probabilistic NeSy approach where neural model predictions are treated as probabilities over variables or facts within logical constraints.

Probabilistic logic constraints are incorporated as a regularization term for the neural network’s loss. Models are typically formulated as deep symbolic parameter models.

- **DeepProbLog (DPL)** [81]: A probabilistic NeSy approach where neural model predictions are treated as probabilities over variables or facts within a probabilistic logic framework. Probabilistic logic constraints are incorporated as a regularization term for the neural network’s loss or as a layer on top of the neural network. Models are typically formulated as deep symbolic parameter models.
- **Logic Tensor Networks (LTNs)** [14]: A fuzzy NeSy approach where neural models represent predicates whose outputs are treated as variables within a fuzzy logic. Fuzzy logic constraints are incorporated as a layer on top of the neural networks. Models are typically formulated as deep symbolic variable models or deep symbolic parameter models.

3.3.1 Semantic Loss (SL)

Semantic loss is a probabilistic NeSy approach where neural network predictions are interpreted as probabilities over propositional variables within logical constraints [138]. It quantifies how well the neural network’s outputs align with the specified constraints, acting as a regularization term to guide the model toward logically consistent predictions by penalizing constraint violations. Typically, this approach creates models inherent to the *neural as symbolic parameter* architectural axioms (Section 2.3.3).

3.3.1.1 Definition and Background

Formally, let $\mathbf{y} = \{y_1, y_2, \dots, y_n\}$ be a set of n propositional variables and a *world* ω is an instantiation of all variables \mathbf{y} , i.e., $\omega = \{y_1 = v_1, \dots, y_n = v_n\}$ for $v_i \in \{0, 1\}$. A world ω satisfies a sentence α , denoted $\omega \models \alpha$, if the sentence evaluates to true in that world. A sentence α *entails* another sentence β , denoted $\alpha \models \beta$, if all worlds that satisfy α also satisfy β .

In semantic loss, each propositional variable $y_i \in \mathbf{y}$ is assigned a probability p_i that it is true. These probabilities $\mathbf{p} = \{p_1, p_2, \dots, p_n\}$ are defined from a neural network. The

semantic loss, denoted as $L_{SL}(\alpha, \mathbf{p})$, measures how closely the neural network's predictions satisfy the sentence α with:

$$L_{SL}(\alpha, \mathbf{p}) \propto -\log \left(\sum_{\omega \models \alpha} \prod_{i: \omega \models y_i} p_i \prod_{i: \omega \models \neg y_i} (1 - p_i) \right)$$

The summation is taken over all worlds ω that satisfy α . The product $\prod_{i: \omega \models y_i} p_i$ measures the probability of all variables assigned true in ω and $\prod_{i: \omega \models \neg y_i} (1 - p_i)$ measures the probability of all variables assigned false in ω . The right side of the equation is the well-known reasoning task weighted model counting [23] that was described earlier in Section 2.3.3.

In practice, semantic loss, weighted by some hyperparameter γ , is applied as a regularization term that encourages the neural model to produce predictions consistent with logical constraints. This allows for flexible integration of symbolic reasoning into neural network training. Formally, this is defined as:

$$L_{\text{total}} = L_{\text{existing}} + \gamma \cdot L_{SL}(\alpha, \mathbf{p}),$$

where L_{existing} is the original loss function.

With the semantic loss defined, let's walk through an example of semantic loss as a mutually exclusive constraint in multi-class classification.

Example 3.3.1. *Consider a multi-class classification problem where exactly one of n possible outcomes should be true. The sentence $\alpha_{\text{exactly_one}}$ enforces that exactly one of the variables y_1, \dots, y_n is true:*

$$\alpha_{\text{exactly_one}} = (y_1 \wedge \neg y_2 \wedge \dots \wedge \neg y_n) \vee (\neg y_1 \wedge y_2 \wedge \dots \wedge \neg y_n) \vee \dots \vee (\neg y_1 \wedge \dots \wedge y_n).$$

The semantic loss for this multi-class setting is computed as:

$$L_{SL}(\alpha_{\text{exactly_one}}, \mathbf{p}) \propto -\log \left(\sum_{i=1}^n p_i \prod_{\substack{j=1 \\ j \neq i}}^n (1 - p_j) \right),$$

where $\mathbf{p} = \{p_1, \dots, p_n\}$ are the probabilities predicted by the neural network for each class.

While, indeed, this shows how more complicated sentences can be incorporated into the problem, it is essential to note that this requires calculating the weighted model count [23]. The exact computation of the weighted-model count is #P-hard. To somewhat circumvent this issue, it is common in the literature to compile logical formulas into a circuit for many inference queries [25, 68, 81]. It is important to note that while this can make inference fast, the compilation step is potentially exponential in time and memory, and there are no guarantees the size of the circuit is not exponential [128]. With all that said, the following section describes how to represent semantic loss functions as NeSy-EBMs using constraints.

3.3.1.2 NeSy-EBM Formulation

Semantic loss can be formulated as a DSPar NeSy-EBM where the propositional variable probabilities are neural network outputs. Since semantic loss does not take as input any observed random variables, then there are no external parameters, i.e., let $\mathbf{x}_{sy} \in \emptyset$ and $\mathbf{w}_{sy} \in \emptyset$. Let g_{nn} be a function with parameters $\mathbf{w}_{nn} \in W_{nn}$ and inputs $\mathbf{x}_{nn} \in X_{nn}$ such that:

$$g_{nn} : W_{nn} \times X_{nn} \mapsto [0, 1]^n,$$

where n is the number of propositional variables involved in the constraints, this function outputs the predicted probabilities:

$$g_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) = [p_i]_{i=1}^n,$$

where p_i represents the predicted probability for the i -th variable.

Define the logical sentence α as a hard constraint $C_H(\omega)$ using an indicator function that represents whether a world ω (an assignment of the propositional variables) satisfies the sentence α :

$$C_H(\omega) = \mathbb{I}(\omega \models \alpha),$$

where $\mathbb{I}(\omega \models \alpha)$ is 1 if the world satisfies the constraint α , and 0 otherwise.

Assuming the weighted model count formulation, the symbolic component of the

NeSy-EBM is a DSPar potential function:

$$\begin{aligned} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) &= \psi_{SL}([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]) \\ &= -\log \left(\sum_{\omega \in \{0,1\}^{|y|}} C_H(\omega) \prod_{i=1}^n p_i^{\omega_i} (1 - p_i)^{1-\omega_i} \right) \end{aligned}$$

where ω_i is the i^{th} propositional variable value.

Given the symbolic potential and variables defined above, the semantic energy function is defined as:

$$\begin{aligned} E_{SL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) &= g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &= -\log \left(\sum_{\omega \in \{0,1\}^{|y|}} C_H(\omega) \prod_{i=1}^n p_i^{\omega_i} (1 - p_i)^{1-\omega_i} \right) \end{aligned} \tag{3.6}$$

3.3.2 DeepProbLog (DPL)

DeepProbLog (DPL) [81], like semantic loss, is a probabilistic NeSy approach that integrates neural network predictions with symbolic reasoning. However, while semantic loss operates over propositional logic, DeepProbLog utilizes a first-order logic framework, allowing for more expressive reasoning capabilities. Specifically, DeepProbLog incorporates neural network outputs as probabilities within the probabilistic programming language ProbLog [39] using neural predicates. This connection to ProbLog gives DeepProbLog access to both probabilistic logic programming and program induction, enabling it to handle more complex symbolic structures and reasoning tasks compared to the propositional-based semantic loss. Typically, this approach creates models inherent to the *neural as symbolic parameter* architectural axioms (Section 2.3.3).

3.3.2.1 Definition and Background

To begin, we review the basics of probabilistic logic programming in ProbLog, following the presentation from [81] (see [39] for further details).

ProbLog: A ProbLog program consists of two main components:

- A set of *probabilistic facts* \mathcal{F} of the form $p :: y$, where p is the probability that the binary target random variable y is true (i.e., $y \in \{0, 1\}$). Note: this can be extended to include a set of observed facts or evidence $p :: x_{sy}$.
- A set \mathcal{R} of *symbolic rules*, which describe how different facts relate to each other.

A subset of the probabilistic facts $F \subseteq \mathcal{F}$ defines a possible instantiation, or *world* ω . This world includes all facts in F and all facts derivable from F using the rules in \mathcal{R} :

$$\omega = F \cup \{y \mid \mathcal{R} \cup F \models y\},$$

where $\mathcal{R} \cup F \models y$ means that the fact y can be derived from the combination of rules \mathcal{R} and facts F . The probability of a world ω is given by the product of the probabilities of the facts in that world:

$$P(\omega) := \prod_{y_i \in F} p_i \prod_{y_i \in \mathcal{F} \setminus F} (1 - p_i),$$

where p_i is the probability assigned to fact y_i .

Example 3.3.2. Consider the following ProbLog program, which models the likelihood of a burglary or earthquake causing an alarm:

Probabilistic Facts:

0.1 :: burglary, 0.2 :: earthquake,

0.5 :: hearsAlarm(mary), 0.4 :: hearsAlarm(john).

Rules:

alarm : – earthquake.

alarm : – burglary.

calls(X) : – alarm, hearsAlarm(X).

Now, consider the subset $F = \{\text{burglary}, \text{hearsAlarm}(\text{mary})\}$ of probabilistic facts. The corresponding world ω includes the derived facts:

$$\omega = \{\text{burglary}, \text{hearsAlarm}(\text{mary}), \text{alarm}, \text{calls}(\text{mary})\}.$$

The probability of this world is:

$$P(\omega) = 0.1 \cdot 0.5 \cdot (1 - 0.2) \cdot (1 - 0.4) = 0.024.$$

DeepProbLog: A DeepProbLog program extends the syntax and semantics of ProbLog by introducing neural predicates, allowing the specification of probabilistic facts based on neural network outputs [81]. Specifically, DeepProbLog introduces neural annotated disjunctions (nADs), which integrate neural network predictions directly into the logic. A neural annotated disjunction is specified as:

$$nn(m_{g_{nn}}, \mathbf{x}_{nn}, u_1) :: h(\mathbf{x}_{nn}, u_1); \dots; nn(m_{g_{nn}}, \mathbf{x}_{nn}, u_n) :: h(\mathbf{x}_{nn}, u_n) \models b_1, \dots, b_m,$$

where \mathbf{x}_{nn} is a vector of features accessible to the neural component identified by $m_{g_{nn}}$. The terms u_1, \dots, u_n represent the possible outputs of the neural network, and the atoms b_1, \dots, b_m are logical conditions. The output of the neural network, $nn(m_{g_{nn}}, \mathbf{x}_{nn}, u_i)$, is interpreted as the probability that the atom $h(\mathbf{x}_{nn}, u_i)$ is true, and the sum of the neural model’s outputs must satisfy:

$$\sum_{i=1}^n nn(m_{g_{nn}}, \mathbf{x}_{nn}, u_i) = 1.$$

The meaning of the nAD is that whenever all the atoms b_1, \dots, b_m hold true, each $h(\mathbf{x}_{nn}, u_i)$ becomes true with probability $nn(m_{g_{nn}}, \mathbf{x}_{nn}, u_i)$.

As we have seen, DeepProbLog shares similarities with semantic loss, but it differs in requiring the neural network’s output to represent a joint probability distribution over the neural outputs for a single example. During inference and learning, DeepProbLog often assumes conditional independence between multiple neural variables. The exact computation of the weighted model count is #P-hard, making inference computationally expensive. To address this, a common approach in the literature, as with semantic loss, is

to compile logical formulas into circuits for more efficient inference across multiple queries [25, 68, 81]. The following section will explain how DeepProbLog can be represented as NeSy-EBMs using constraints.

3.3.2.2 NeSy-EBM Formulation

DeepProbLog can be formulated as a DSPar NeSy-EBM where the fact probabilities are defined with both the symbolic parameters and the neural network outputs. Let \mathbf{x}_{sy} be the observed random variables (evidence), \mathbf{y} be the target random variables (facts), and \mathbf{w}_{sy} be symbolic parameters over facts not parameterized by a neural network (probabilities). Let g_{nn} be a function with parameters $\mathbf{w}_{nn} \in W_{nn}$ and inputs $\mathbf{x}_{nn} \in X_{nn}$ such that:

$$g_{nn} : W_{nn} \times X_{nn} \mapsto [0, 1]^n,$$

where n is the number of propositional variables involved in the constraints. Without loss of generality the fact probabilities, \mathbf{p} , are partitioned into symbolic parameters and these neural network outputs:

$$\mathbf{p} = \begin{bmatrix} \mathbf{w}_{sy} \\ \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) \end{bmatrix}$$

where p_i represents the predicted probability for the i -th variable.

The probability of a world ω , defined by a subset of probabilistic facts $F \subseteq \mathcal{F}$, is a function of the fact probabilities \mathbf{p} , and therefore a function of \mathbf{w}_{sy} and the neural network outputs $\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$:

$$P_{\omega}(\mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = \prod_{\mathbf{w}_{sy}^j \in F} \mathbf{w}_{sy}^j \prod_{\mathbf{w}_{sy}^j \in \mathcal{F} \setminus F} (1 - \mathbf{w}_{sy}^j) \prod_{\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})^j \in F} \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})^j \prod_{\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})^j \in \mathcal{F} \setminus F} (1 - \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})^j)$$

Finally, unlike semantic loss, DeepProbLog typically evaluates probabilities with respect to *queries*. A *query* is a symbolic atom q whose probability we want to compute based on the probabilistic facts and the neural network outputs. For example, the marginal probability of a query atom q is computed by summing over the probabilities of all worlds ω in which q is true.

Let $C_H(\omega, q)$ be the constraint function, which acts as an indicator function returning 1 if the world ω satisfies the condition that the query atom q is true:

$$C_H(\omega, q) = \begin{cases} 1 & \text{if } q \in \omega \\ 0 & \text{otherwise.} \end{cases}$$

Assuming the weighted model count formulation, the symbolic component of the NeSy-EBM is a DSPar potential function for a single query q is defined as:

$$\begin{aligned} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) &= \psi_{DPL}^q([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]) \\ &= d\left(q, \sum_{\omega \in \{0,1\}^{|\mathbf{y}|}} C_H(\omega, q) P_{\omega}(\mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))\right) \end{aligned}$$

Where $d(\cdot, \cdot)$ is the distance between the predicted probability of the query and its true probability. Now, the energy function is defined over a sum of all queries \mathbf{q} .

$$E_{DPL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \sum_{i=1}^{|\mathbf{q}|} \psi_{DPL}^{q_i}([\mathbf{y}, \mathbf{x}_{sy}], [\mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]).$$

3.3.3 Logic Tensor Networks (LTNs)

Logic Tensor Networks (LTNs) [14] are a fuzzy Neural-Symbolic Energy-Based Model (NeSy-EBM) approach, integrating neural network predictions with logic-based reasoning. In LTNs, neural networks provide real-valued truth values for predicates, which are then manipulated using fuzzy logic operations to evaluate logical formulae. The satisfaction levels of the logical formulae are aggregated through generalized mean semantics, which form the basis of the energy function. Typically, this approach creates models inherent to the *neural as symbolic variable* architectural axioms (Section 2.3.4).

LTNs use product real logic operators to define fuzzy truth values for logical connectives:

$$\neg(a) := 1 - a, \quad \wedge(a, b) := a \cdot b, \quad \vee(a, b) := a + b - a \cdot b, \quad \implies (a, b) := a + b - a \cdot b.$$

Additionally, LTNs use formula aggregators, such as generalized mean semantics, to handle existential and universal quantifiers over collections of truth values, denoted by $\mathbf{a} = [a_i]_{i=1}^n$:

$$\exists(\mathbf{a}) := \left(\frac{1}{n} \sum_{i=1}^n a_i^p \right)^{\frac{1}{p}}, \quad \forall(\mathbf{a}) := 1 - \left(\frac{1}{n} \sum_{i=1}^n (1 - a_i)^p \right)^{\frac{1}{p}},$$

where $p \in \mathbb{R}_+$ is a hyperparameter controlling the smoothness of the quantifiers.

In LTNs, neural networks instantiate predicates with values from $[0, 1]$, representing the degree to which a predicate is satisfied. For example, given two entities u and v , the predicate $P(u, v)$ can be defined as the output of a neural network $g_{nn}(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w}_{nn})$, which maps the feature vectors $\mathbf{X}[u]$ and $\mathbf{X}[v]$ to a truth value in $[0, 1]$.

Example 3.3.3. Consider the following logical formula, which expresses that for each entity u , there exists some entity v such that both predicates $P(u, v)$ and $Q(v)$ hold true:

$$\exists v \in \mathcal{V} (P(u, v) \wedge Q(v)).$$

Let $\mathbf{X}_{\mathcal{U}}$ and $\mathbf{X}_{\mathcal{V}}$ represent the feature vectors for the sets of entities \mathcal{U} and \mathcal{V} , respectively. The predicates $P(u, v)$ and $Q(v)$ can be instantiated with neural network outputs: - $P(u, v)$ is given by the neural network $g_{nn}(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w}_{nn})$, - $Q(v)$ could be a constant truth value or another neural network prediction.

Using the generalized mean semantics for the existential quantifier, we define the real-valued logic function for the above formula as:

$$h_u(\mathbf{X}_{\mathcal{U}}, \mathbf{X}_{\mathcal{V}}, \mathbf{x}_Q; \mathbf{w}_{nn}) = \left(\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} (g_{nn}(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w}_{nn}) \cdot \mathbf{x}_Q[v])^p \right)^{\frac{1}{p}},$$

where $\mathbf{x}_Q[v]$ is the truth value for the predicate $Q(v)$, and $g_{nn}(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w}_{nn})$ is the neural network output for the predicate $P(u, v)$.

The satisfaction level of the formula for all entities u is then aggregated using the universal quantifier:

$$G(\mathbf{w}_{nn}) = 1 - \left(\frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} (1 - h_u(\mathbf{X}_{\mathcal{U}}, \mathbf{X}_{\mathcal{V}}, \mathbf{x}_Q; \mathbf{w}_{nn}))^p \right)^{\frac{1}{p}}.$$

This example illustrates how LTNs leverage neural networks to assign fuzzy truth values to predicates and apply these values in evaluating logical formulas. The next section will explain how LTNs can be represented as NeSy-EBMs using symbolic constraints.

3.3.3.1 NeSy-EBM Formulation

LTNs can be formulated as a DSVar NeSy-EBM, where the satisfaction of symbolic constraints is driven by neural network outputs. Let \mathbf{x}_{sy} be the observed random variables (constants), and since the symbolic component does not have trainable parameters, define $\mathbf{w}_{sy} \in \emptyset$. Let g_{nn} be a function with parameters $\mathbf{w}_{nn} \in W_{nn}$ and inputs $\mathbf{x}_{nn} \in X_{nn}$ such that:

$$g_{nn} : W_{nn} \times X_{nn} \mapsto [0, 1]^n,$$

where n is the number of variables involved in the constraints.

Define $C_S^{Agg}(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy})$ as the soft constraint function that takes as input the output of a set of neural networks $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ and applies the collection of aggregation functions Agg in some way that maintains differentiability (e.g., the generalized mean or quantifiers).

The symbolic component of the NeSy-EBM is a DSVar potential function:

$$\begin{aligned} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) &= \psi_{LTN}([\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})], [\mathbf{w}_{sy}]) \\ &= C_S^{Agg}(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy}) \end{aligned}$$

Given the symbolic potential and variables defined above, the energy function for LTNs is defined as:

$$\begin{aligned} E_{LTN}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) &= g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &= C_S^{Agg}(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy}). \end{aligned}$$

Part III

Neural-Symbolic Design Principles

Chapter 4

Neural Symbolic Inference and Learning

The previous two chapters established the foundational components of neural-symbolic approaches, providing a set of architectural axioms and a universal mathematical language through neural-symbolic energy-based models. In this chapter, I shift focus to begin studying principled designed decisions, starting with NeSy inference and learning: First, I define NeSy inference within the NeSy-EBM framework, outlining typical inference tasks frequently encountered in machine learning, such as prediction, classification, ranking, and density estimation. While the formal structure of NeSy-EBM inference provides a robust theoretical foundation, I categorize three practical strategies: computation graph vs. optimization execution, instance vs. global model construction, and decomposed vs. unified task structure. Following this, I introduce the general formulation of NeSy-EBM learning, which details the mechanisms for jointly training these models. As with inference, I categorize typical learning approaches and offer insight into their application.

Although this chapter lays out the core design principles of inference and learning, it is essential to recognize that these formulations do not yet tackle the practical challenges associated with their implementation. Blindly applying NeSy inference and learning techniques without regard to a task’s specific requirements and nuances can result in suboptimal performance or unintended outcomes. Therefore, this chapter serves as a foundation for understanding the intricacies of reasoning and training NeSy approaches, with a deeper exploration of potential pitfalls and challenges in the following chapter (Chapter 5).

4.1 NeSy-EBM Inference

NeSy-EBMs provide a flexible and general framework capable of handling various inference and reasoning tasks. Formally, given inputs and parameters $(\mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn}$, the energy function $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ serves as the basis for defining a variety of inference tasks, including but not limited to:

- **Prediction, Classification, and Decision Making:** These tasks aim to find target variables that minimize the energy function, corresponding to selecting the most probable or optimal outcome given the inputs.

$$\hat{\mathbf{y}} = \arg \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}). \quad (4.1)$$

- **Ranking:** This task aims to order a set of target variables based on their energy values, with lower energy indicating a more favorable or probable outcome.

$$E(\mathbf{y}^{\mathbf{r}^1}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \leq \dots \leq E(\mathbf{y}^{\mathbf{r}^p}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}). \quad (4.2)$$

- **Detection:** This task aims to determine whether a target variable \mathbf{y} satisfies a predefined energy threshold τ , often used in classification or anomaly detection.

$$D(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}; \tau) = \begin{cases} 1 & \text{if } E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \leq \tau, \\ 0 & \text{otherwise.} \end{cases} \quad (4.3)$$

- **Density Estimation:** This task involves estimating the conditional probability distribution of a target variable \mathbf{y} , where the energy function defines a probability density, such as a Gibbs distribution:

$$P(\mathbf{y} | \mathbf{x}_{sy}, \mathbf{x}_{nn}; \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \frac{e^{-\beta E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}} e^{-\beta E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})}}, \quad (4.4)$$

where β is a positive inverse temperature parameter controlling the sharpness of the distribution.

- **Generation:** In these tasks, the objective is to sample target variable states from a probability distribution defined by the energy function:

$$\mathbf{y} \sim P(\mathbf{y}|\mathbf{x}_{sy}, \mathbf{x}_{nn}; \mathbf{w}_{sy}, \mathbf{w}_{nn}). \quad (4.5)$$

The aforementioned tasks represent just a subset of the inference problems that can be formulated within the NeSy-EBM framework. These examples illustrate how the energy function can be leveraged for both deterministic and probabilistic reasoning, depending on the specific requirements of the task at hand. In the following section, I will provide a comprehensive guide to typical inference and reasoning pipelines within the field of NeSy.

4.2 NeSy Learning Design Principles

While the previous section introduced the general formulation of machine learning tasks for NeSy-EBMs, the practical implementation of these systems varies significantly across the field of NeSy. This variability stems from the inherent flexibility and broad scope of NeSy methods, allowing for a wide range of interpretations and adaptations, particularly in the symbolic components. While numerous incremental improvements have been made to various inference settings, in this section, I propose three key inference designs:

- **Computation Graph vs. Optimization Execution** (Section 4.2.1): Computation graph execution uses a fixed, directed acyclic graph to propagate reasoning, similar to neural network layers. Optimization execution, by contrast, iteratively minimizes or maximizes an energy or loss function, typically adjusting neural and symbolic variables to satisfy the constraints.
- **Instance vs. Global Model Construction** (Section 4.2.2): Instance model construction dynamically generates structures for each query or example, tailoring the inference path to the specific input. Global models, by contrast, rely on a shared, reusable structure for all examples.

- **Decomposed vs. Unified Task Structure** (Section 4.2.3): Decomposed task structure breaks down into smaller sub-tasks, where neural models handle lower-level operations and symbolic models perform higher-level reasoning. Unified task structures, by contrast, involve neural and symbolic components working together on the same task, such as using symbolic reasoning to ensure the neural model’s outputs adhere to constraints.

It is important to emphasize that this discussion centers on the design of NeSy systems. The specific details of the inference processes—such as techniques for optimization reasoning or algorithmic constructions of instance-based models—are highly diverse and will not be covered exhaustively in this dissertation. For more fine-grained categorizations, I defer to existing taxonomies and approaches that delve deeper into these nuanced distinctions [32, 43, 45, 87].

4.2.1 Computation Graph vs. Optimization Execution

One of the most fundamental distinctions in neural-symbolic systems lies in how inference is executed, specifically whether reasoning is driven by a computation graph or an optimization process. Similar distinctions are observed in other fields, such as the differences between undirected and directed graphical models [69], between proof-based and model-based logical methods [87], etc. In the context of NeSy-EBMs, this distinction is reflected in how the system’s energy function is processed during inference.

Computation Graph-Based Execution: In computation graph-based execution, the reasoning process is represented as a sequence of computations, where each step applies a transformation to the inputs. This approach draws similarities to the way in which neural networks perform execution, with each layer applying a specific operation.

Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} be the observed symbolic variables, \mathbf{x}_{nn} be the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} be the symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model with parameters \mathbf{w}_{nn} and input \mathbf{x}_{nn} . In computation graph-based execution, the reasoning process can be expressed as a

function f , where each step in the computation graph applies a transformation to these inputs:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \equiv f(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})),$$

where f denotes the sequence of transformations applied to the symbolic inputs \mathbf{x}_{sy} , neural outputs $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ and target variables \mathbf{y} . This method of execution is particularly suited for NeSy methods that rely on differentiable proof-based reasoning [87], where symbolic rules or constraints guide the structure of the neural model. For example, differentiable theorem provers and neural logic programming systems commonly leverage computation graph-based execution, as the structure of the inference process can be naturally encoded as a graph of operations.

Optimization-Based Execution: In optimization-based execution, the reasoning process is framed as an optimization problem where the goal is to find an optimal configuration of variables that minimizes or maximizes an energy or loss function representing the system’s overall state. Unlike computation graph-based execution, which involves a fixed sequence of transformations, optimization-based execution adjusts the target variables to satisfy symbolic constraints.

Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} the observed symbolic inputs, and \mathbf{x}_{nn} the neural inputs, with parameters \mathbf{w}_{sy} for the symbolic model and \mathbf{w}_{nn} for the neural model. Obviously, this depends on the inference task, but for prediction, classification, and decision-making, this looks like this:

$$\hat{\mathbf{y}} = \arg \min_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}),$$

where E is the energy function combining the symbolic and neural components of the model. This type of execution often relies on gradient-based techniques, such as gradient descent, or more specialized solvers depending on the form of the energy function and constraints. Optimization-based execution is commonly employed in neural-symbolic systems that require constraint satisfaction, probabilistic inference, or structured prediction tasks.

4.2.2 Instance vs. Global Model Construction

Another critical design choice in neural-symbolic systems is how the neural-symbolic model is constructed. Symbolic systems often produce potentially large grounded models, making it advantageous in some inference problems to ground only a subset of the graph—the portion relevant to the specific query or problem. This distinction gives rise to two primary inference approaches: *instance* model construction and *global* model construction. For reference, not only is this choice important to NeSy systems, but it is also prevalent across purely symbolic literature. For example, two of the most prominent frameworks, ProbLog [39] and PSL [11], exemplify these approaches, with ProbLog typically employing instance-based model construction and PSL favoring global model construction.

Example 4.2.1 (Instance v.s. Global Model). *Consider a purely symbolic propositional logic scenario with two rules: $A \rightarrow C$ and $B \rightarrow C$. The goal is to predict the value of C given an input.*

In the instance model approach, the system dynamically selects and grounds only the relevant symbolic rules for each specific input. For example, if the input is $A = 1$, the system selects the rule $A \rightarrow C$ and ignores the rule $B \rightarrow C$, as B is irrelevant to this specific instance. The system then constructs a reasoning process based on the selected rule, effectively transforming it into a computation graph. By following this reasoning path, it can directly infer that C must also be true ($C = 1$).

In contrast, in the global model approach, all rules are considered simultaneously, forming a universal model for all potential inputs. Given the same input $A = 1$, the system treats this scenario as an optimization problem, searching for a configuration of values that satisfies the entire set of rules. For example, the global model may explore assignments such as $A = 1, B = 0, C = 1$ or $A = 1, B = 1, C = 1$.

While this is a purely symbolic example, it is simple to extend to neural-symbolic scenarios, such as when the value of A is determined by a neural model. Furthermore, execution strategies are not strictly tied to the model constructed. For instance, an instance model construction can involve optimization, while a global model can also be used to build a computation graph.

Instance Model Construction: Instance model construction dynamically generates symbolic structures tailored to each specific input, allowing the reasoning process to adapt to the unique characteristics of each example. For instance, in the MNIST-Add example presented in the architecture chapter (Example 2.1.1), an instance-based model grounds only the additions corresponding to the most probable digit classes instead of grounding every possible two-digit addition. This approach reduces the number of possible groundings from $10 \times 10 = 100$ to just $1 \times 1 = 1$. While in practice, the model typically constructs symbolic structures dynamically during inference, within the framework of NeSy-EBMs, the energy function must account for all potential symbolic structures.

Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} the observed symbolic inputs, and \mathbf{x}_{nn} the neural inputs, with parameters \mathbf{w}_{sy} for the symbolic model and \mathbf{w}_{nn} for the neural model. Let \mathcal{J} denote the finite set of all possible symbolic structures. The energy function E can then be expressed as a summation of the individual energy contributions, with structures not relevant to a specific example (\mathbf{x}_{nn} and \mathbf{x}_{sy}) assigned an energy of zero:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \sum_{\mathcal{M} \in \mathcal{J}} E_{\mathcal{M}}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}),$$

where $E_{\mathcal{M}}$ denotes the energy function associated with a symbolic structure \mathcal{M} . Each instance-specific energy function $E_{\mathcal{M}}$ is defined as:

$$E_{\mathcal{M}}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} \mathcal{M}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) & \text{if } \mathcal{M} \text{ is the valid structure,} \\ 0 & \text{otherwise.} \end{cases}$$

The definition of a valid symbolic structure depends on the specific context or task being addressed by the model, allowing the system to adapt its reasoning dynamically to each instance. For example, in the MNIST-Add task, all additions outside of the most probable digit classes are “invalid” symbolic structures for that energy value. In practice, a single symbolic structure is typically generated based on the given input, while all other potential structures are assumed to contribute zero to the total energy function.

Global Model Construction: In contrast, global model construction utilizes a shared symbolic structure that applies uniformly across all examples. This approach encapsulates

a consistent set of symbolic constraints that remain unchanged regardless of the specific input instance.

Formally, let \mathbf{y} be the target random variables, \mathbf{x}_{sy} the observed symbolic inputs, and \mathbf{x}_{nn} the neural inputs, with parameters \mathbf{w}_{sy} for the symbolic model and \mathbf{w}_{nn} for the neural model. In this case, let $\mathcal{M}_{\text{global}}$ denote the shared symbolic model. The corresponding energy function is given by:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = E_{\mathcal{M}_{\text{global}}}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}),$$

4.2.3 Decomposed vs. Unified Task Structure

The final design decision I highlight revolves around two common structural approaches for addressing different types of neural-symbolic problems. These approaches are typically motivated by distinct use cases: (1) amalgamating neural predictions through symbolic reasoning or (2) refining a neural model’s predictions using symbolic guidance. This distinction gives rise to an architectural design dichotomy: *decomposed task structures* and *unified task structures*. Again, similar to the other design decisions, this has been seen throughout many research areas, such as the Statistical Relational Learning (SRL) community [56].

Decomposed Task Structure In a *decomposed task structure*, the neural network’s outputs predict intermediate concepts that are distinct from the target variables, i.e., the outputs do not directly map to the domain of the target variables. Instead, the neural network predicts auxiliary concepts that the symbolic model processes to infer the final target variables. Typically, the neural model is responsible for lower-level tasks, such as perception or feature extraction, while the symbolic model handles higher-level reasoning or decision-making tasks based on these neural outputs.

Formally, let \mathbf{y} denote the target random variables, \mathbf{x}_{sy} the observed symbolic variables, \mathbf{x}_{nn} the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} the symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model with parameters \mathbf{w}_{nn} and input \mathbf{x}_{nn} . A *decomposed task structure* is defined such that the concepts, \mathbf{c} , predicted by

the neural model do not belong to the same domain of the target variables:

$$\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) = \mathbf{c}, \quad \mathbf{c} \in \mathcal{C} \text{ and } \mathcal{C} \not\equiv \mathcal{Y}.$$

Consider the task of *MNIST Addition* introduced in the architecture chapter (Example 2.1.1). In this scenario, the neural model predicts the individual digits in the given addition problem, which are not directly part of the target domain of additions. The symbolic model then performs an arithmetic sum of these recognized digits to produce the final target variable. Here, each component addresses a distinct task: the neural model predicts digit classes, while the symbolic model computes their sum.

Unified Task Structure In contrast, in a *unified task structure*, the neural and symbolic components work collaboratively to predict the same target variables, i.e., the outputs directly map to the domain of the target variables. Typically, the neural model makes an initial prediction, while the symbolic model adjusts these predictions based on the problem's constraints.

Formally, let \mathbf{y} denote the target random variables, \mathbf{x}_{sy} the observed symbolic variables, \mathbf{x}_{nn} the observed neural inputs, and \mathbf{w}_{sy} and \mathbf{w}_{nn} the symbolic and neural parameters, respectively. Let \mathbf{g}_{nn} represent a neural model with parameters \mathbf{w}_{nn} and input \mathbf{x}_{nn} . A *unified task structure* is defined such that the concepts \mathbf{c} , predicted by the neural model, belong to the same domain as the target variables:

$$\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) = \mathbf{c}, \quad \mathbf{c} \in \mathcal{C} \text{ and } \mathcal{C} \equiv \mathcal{Y}.$$

Consider the task of *visual Sudoku puzzle classification* introduced in the modeling patterns section (Example 3.2.1). In this scenario, the neural model classifies the digits in a given visual Sudoku board, directly predicting values that belong to the target domain. The symbolic model then refines these predictions by enforcing Sudoku constraints, ensuring that no digit is repeated within any row, column, or sub-grid. Here, both the neural and symbolic components contribute to the same task, predicting digits.

4.3 NeSy-EBM Learning

Having identified a range of modeling paradigms and inference paradigms, let's now turn our attention to learning. This section formalizes the NeSy-EBM learning problem, introduces a general loss function, and discusses several algorithms for addressing these challenges. While I contributed to the high-level conceptualization and helped integrate these ideas into the broader NeSy framework, the detailed technical work and proofs are largely the contributions of my colleague Charles. For a more comprehensive and detailed discussion, I refer the reader to their work [43]. My role primarily involved helping to frame these concepts within the larger context of NeSy-EBMs.

At a high level, NeSy-EBM learning involves determining the weights of an energy function that assigns higher compatibility (lower energy) to targets and neural outputs that align closely with the true labels from the training data. However, predicting with NeSy-EBMs requires solving a complex mathematical program, which presents several challenges for learning. For example, NeSy-EBM predictions may not be differentiable with respect to the model parameters, making the direct application of automatic differentiation either impractical or ineffective. Even in cases where predictions are differentiable, the associated gradients often rely on properties of the energy function at its minimizer, which can be computationally expensive to compute. This section introduces general and principled learning frameworks for NeSy-EBMs that address these challenges.

This section is organized as follows: First, Section 4.3.1 will introduce some preliminary notation and the general definition of NeSy-EBM learning. Then, Section 4.3.2 will present four learning losses that define the learning problem. Finally, Section 4.3.3 will categorize four learning algorithms.

4.3.1 Definition

The following notation and general definition of NeSy-EBM learning are used throughout this dissertation. The training dataset, denoted by \mathcal{S} , is comprised of P samples and indexed by $\{1, \dots, P\}$. Each sample, \mathcal{S}_i where $i \in \{1, \dots, P\}$, is a tuple of *inputs*, *labels*, and *latent variable domains*. Sample *inputs* consist of neural inputs, \mathbf{x}_{nn}^i from \mathcal{X}_{nn} ,

and symbolic inputs, \mathbf{x}_{sy}^i from \mathcal{X}_{sy} . Similarly, sample *labels* consist of neural and symbolic labels, which are truth values corresponding to a subset of the neural predictions and target variables, respectively. Neural labels, denoted by \mathbf{t}_{nn}^i , are $d_{nn}^i \leq d_{nn}$ dimensional real vectors from a domain \mathcal{T}_{nn}^i , i.e., $\mathbf{t}_{nn}^i \in \mathcal{T}_{nn}^i \subseteq \mathbb{R}^{d_{nn}^i}$. Target labels, denoted by $\mathbf{t}_{\mathcal{Y}}^i$, are from a domain $\mathcal{T}_{\mathcal{Y}}^i$ that is a $d_{\mathcal{T}_{\mathcal{Y}}^i} \leq d_{\mathcal{Y}}$ subspace of the target domain \mathcal{Y} , i.e., $\mathbf{t}_{\mathcal{Y}}^i \in \mathcal{T}_{\mathcal{Y}}^i$. Lastly, the neural and symbolic *latent variable domains* are subspaces of the range of the neural component and the target domain, respectively, corresponding to the set of unlabeled variables. The range of the neural component, $\mathbb{R}^{d_{nn}}$, is a superset of the Cartesian product of the neural latent variable domain, denoted by \mathcal{Z}_{nn}^i , and \mathcal{T}_{nn}^i , i.e., $\mathbb{R}^{d_{nn}} \supseteq \mathcal{T}_{nn}^i \times \mathcal{Z}_{nn}^i$. Similarly, the target domain \mathcal{Y} is a superset of the Cartesian product of the latent variable domain, denoted by $\mathcal{Z}_{\mathcal{Y}}^i$, and $\mathcal{T}_{\mathcal{Y}}^i$, i.e., $\mathcal{Y} \supseteq \mathcal{T}_{\mathcal{Y}}^i \times \mathcal{Z}_{\mathcal{Y}}^i$. With this notation, the training dataset is expressed as follows:

$$\mathcal{S} := \{(\mathbf{t}_{\mathcal{Y}}^1, \mathbf{t}_{nn}^1, \mathcal{Z}_{nn}^1, \mathcal{Z}_{\mathcal{Y}}^1, \mathbf{x}_{sy}^1, \mathbf{x}_{nn}^1), \dots, (\mathbf{t}_{\mathcal{Y}}^P, \mathbf{t}_{nn}^P, \mathcal{Z}_{nn}^P, \mathcal{Z}_{\mathcal{Y}}^P, \mathbf{x}_{sy}^P, \mathbf{x}_{nn}^P)\}. \quad (4.6)$$

A learning objective, denoted by \mathcal{L} , is a functional that maps an energy function and a training dataset to a scalar value. Formally, let \mathcal{E} be a family of energy functions indexed by weights from $\mathcal{W}_{sy} \times \mathcal{W}_{nn}$:

$$\mathcal{E} := \{E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \mid (\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}\}. \quad (4.7)$$

Then, a learning objective is the function:

$$\mathcal{L} : \mathcal{E} \times \{\mathcal{S}\} \rightarrow \mathbb{R}. \quad (4.8)$$

Learning objectives follow the standard empirical risk minimization framework and are separable over elements of \mathcal{S} as a sum of *per-sample loss functionals* denoted by L^i for each $i \in \{1, \dots, P\}$. A loss functional for the sample $\mathcal{S}_i \in \mathcal{S}$ is the function:

$$L^i : \mathcal{E} \times \{\mathcal{S}_i\} \rightarrow \mathbb{R}. \quad (4.9)$$

A regularizer, denoted by $\mathcal{R} : \mathcal{W}_{sy} \times \mathcal{W}_{nn} \rightarrow \mathbb{R}$, is added to the learning objective and NeSy-EBM learning is the following minimization problem:

$$\begin{aligned} & \arg \min_{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}} \mathcal{L}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \\ &= \arg \min_{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}} \frac{1}{P} \sum_{i=1}^P L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}). \end{aligned} \quad (4.10)$$

4.3.2 Learning Losses

A NeSy-EBM learning loss functional, L^i , can be decomposed into three components: *neural*, *value-based*, and *minimizer-based* losses. This subsection provides an intuitive overview of each component. For more details and formal proofs, which are both primarily contributions of my colleague, I refer the reader to their fantastic work [43].

At a high level, the neural loss measures the quality of the neural component independent from the symbolic component. Then, the value-based and minimizer-based losses measure the quality of the NeSy-EBM as a whole. Moreover, value-based and minimizer-based losses are functionals mapping a parameterized energy function and a training sample to a real value and are denoted by $L_{Val} : \mathcal{E} \times \mathcal{S} \rightarrow \mathbb{R}$ and $L_{Min} : \mathcal{E} \times \mathcal{S} \rightarrow \mathbb{R}$, respectively. The learning loss components are aggregated via summation:

$$\begin{aligned} & L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \\ &= L_{NN}(\mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{t}_{nn}^i) && \text{Neural} \\ & \quad + L_{Val}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) && \text{Value-Based} \\ & \quad + L_{Min}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) && \text{Minimizer-Based} \end{aligned} \quad (4.11)$$

4.3.2.1 Neural Learning Losses

Neural learning losses are scalar functions of the neural network output and the neural labels and are denoted by $L_{NN} : \text{Range}(\mathbf{g}_{nn}) \times \mathcal{T}_{nn}^i \rightarrow \mathbb{R}$. For example, a neural learning loss may be the familiar binary cross-entropy loss applied in many categorical prediction settings. Minimizing a neural learning loss with respect to neural component parameters is achievable via backpropagation and standard gradient-based algorithms.

4.3.2.2 Value-Based Learning Losses

Value-based learning losses depend on the model weights strictly via minimizing values of an objective defined with the energy. More formally, denote an objective function by f , which maps a compatibility score, target variables, and the training sample to a scalar value:

$$f : \mathbb{R} \times \mathcal{Y} \times \{\mathcal{S}_i\} \rightarrow \mathbb{R}. \quad (4.12)$$

An *optimal value-function*, denoted by V , is the value of f composed with the energy function and minimized over the target variables:

$$\begin{aligned} V(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \hat{\mathbf{y}}, \mathcal{S}_i) \\ &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} f(g_{sy}(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn})), \hat{\mathbf{y}}, \mathcal{S}_i) \end{aligned} \quad (4.13)$$

Value-based learning losses are functions of one or more optimal value functions. This work considers three instances of optimal value functions: 1) *latent*, $V_{\mathcal{Z}}$, 2) *full*, $V_{\mathcal{Y}}$, 3) and *convolutional*, V_{conv} . The latent optimal value function is the minimizing value of the energy over the latent targets. Further, the labeled targets are fixed to their true values using the following indicator function:

$$I_{\mathcal{T}_{\mathcal{Y}}^i}(\mathbf{y}, \mathbf{t}_{\mathcal{Y}}^i) := \begin{cases} 0 & \mathbf{y} = \mathbf{t}_{\mathcal{Y}}^i \\ \infty & \text{o.w.} \end{cases}. \quad (4.14)$$

The full optimal value function is the minimizing value of the energy over all of the targets. Lastly, the convolutional optimal value function is the infimal convolution of the energy

function and a function $d : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathcal{R}$ scaled by a positive real value $\lambda \in \mathcal{R}$. Formally:

$$\begin{aligned} V_Z(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &:= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + I_{\mathcal{T}_Y^i}(\hat{\mathbf{y}}, \mathbf{t}_Y^i), \\ &= \min_{\hat{\mathbf{z}} \in \mathcal{Z}_Y^i} E((\mathbf{t}_Y^i, \hat{\mathbf{z}}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \quad \text{latent} \end{aligned} \quad (4.15)$$

$$V_Y(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) := \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \quad \text{full} \quad (4.16)$$

$$V_{conv}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i; \mathbf{y}, \lambda) := \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + \lambda \cdot d(\hat{\mathbf{y}}, \mathbf{y}). \quad \text{convolutional} \quad (4.17)$$

Given these optimal value functions, a collection of learning losses can be called upon from previous EBM literature [74]. For instance consider the following two *energy* and *structured perceptron*:

- **Energy Loss:** The simplest value-based learning loss is the *energy loss*, denoted by L_{Energy} . The energy loss is the latent optimal value function,

$$L_{Energy}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) := V_Z(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i). \quad (4.18)$$

Minimizing the energy loss encourages the parameters of the energy function to produce low energies given the observed true values of the input and target variables.

- **Structured Perceptron Loss:** Another simple value-based learning loss is the *Structured Perceptron* loss, denoted by L_{SP} [75, 29]. The structured perception loss is the difference between the latent and full optimal value functions,

$$L_{SP}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) := V_Z(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) - V_Y(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i). \quad (4.19)$$

The gradients of these losses with respect to both the neural and symbolic weights are non-trivial to compute. However, Milgrom and Segal (2002) provides a general theorem for deriving the gradient of optimal value functions with respect to problem parameters, given their existence. Building on this, Dickens (2024) extends and specializes this result to optimal value functions in the context of NeSy-EBMs.

With that said, performance metrics are not always aligned with value-based losses and are known to result in degenerate solutions [74, 103]. For example, without a carefully designed inductive bias, the energy loss in (4.18) may only learn to reduce the energy of all target variables without improving the predictive performance of the NeSy-EBM. One fundamental cause of this issue is that value-based losses are not directly functions of the NeSy-EBM prediction as defined in (4.1), i.e., value-based losses are not functions of an energy minimizer. The following subsection discusses these sets of losses in greater detail.

4.3.2.3 Minimizer-Based Learning Losses

A *minimizer-based* loss is a composition of a differentiable loss, such as cross-entropy or mean squared error, with the energy minimizer. Intuitively, minimizer-based losses penalize parameters yielding predictions distant from the labeled training data. Dickens (2024) shows that for NeSy-EBMs, three assumptions are required for defining this loss and for stable gradient decent learning.

Firstly, to ensure that a minimizer-based loss is well-defined, an assumption of the existence of a unique energy minimizer, denoted by \mathbf{y}^* , for every training sample is made:

Assumption 1. *The energy function is minimized over the targets at a single point for every input and weight and is, therefore, a function:*

$$\begin{aligned} \mathbf{y}^* : \mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn} &\rightarrow \mathcal{Y} \\ (\mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) &\mapsto \arg \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \end{aligned} \quad (4.20)$$

Given this assumption, a minimizer-based loss can be defined over a function d as follows:

$$\begin{aligned} L_{Min}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) &:= d(\arg \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i) \\ &:= d(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i) \end{aligned} \quad (4.21)$$

where $d : \mathcal{Y} \times \mathcal{T}_{\mathbf{y}}^i \rightarrow \mathbb{R}$,

To ensure principled gradient-based learning, an additional assumption must be made that the minimizer is differentiable.

Assumption 2. *The minimizer, \mathbf{y}^* , is differentiable with respect to the weights at every point in $\mathcal{X}_{sy} \times \mathcal{X}_{nn} \times \mathcal{W}_{sy} \times \mathcal{W}_{nn}$.*

Under Assumption 2, the chain rule of differentiation yields the gradient of a minimizer-based loss with respect to the neural and symbolic weights:

$$\begin{aligned} & \nabla_{\mathbf{w}_{sy}} L_{Min}(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i) \\ &= \nabla_3 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})^T \nabla_1 d(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i), \end{aligned} \quad (4.22)$$

$$\begin{aligned} & \nabla_{\mathbf{w}_{nn}} L_{Min}(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i) \\ &= \nabla_4 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})^T \nabla_1 d(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i), \end{aligned} \quad (4.23)$$

where $\nabla_3 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ and $\nabla_4 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ are the Jacobian matrices of the unique energy minimizer with respect to the third and fourth arguments of \mathbf{y}^* , the symbolic and neural weights, respectively, and $\nabla_1 d(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{t}_{\mathbf{y}}^i)$ is the gradient of the supervised loss with respect to its first argument.

A primary challenge of minimizer-based learning is computing the Jacobian matrices of partial derivatives, $\nabla_3 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ and $\nabla_4 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$. To derive explicit expressions for them typically demands the following additional assumption on the continuity properties of the energy function.

Assumption 3. *The energy, E , is twice differentiable with respect to the targets at the minimizer, \mathbf{y}^* , and the Hessian matrix of second-order partial derivatives with respect to the targets, $\nabla_{1,1} E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$, is invertible. Further, the minimizer is the unique target satisfying first-order conditions of optimality, i.e.,*

$$\forall \mathbf{y} \in \mathcal{Y}, \quad \nabla_1 E(\mathbf{y}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = 0 \iff \mathbf{y} = \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \quad (4.24)$$

Assumption 3 is satisfied by energy functions that are, for instance, smooth and strongly convex in the targets. Under Assumption 3, the first-order optimality condition

establishes the minimizer as an implicit function of the weights, and implicit differentiation yields the following equalities:

$$\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \nabla_3 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \quad (4.25)$$

$$= -\nabla_{1,4}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$$

$$\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \nabla_4 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \quad (4.26)$$

$$= -\nabla_{1,5}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn})$$

Solving for the Jacobians of the minimizer:

$$\nabla_3 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = -(\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}))^{-1} \quad (4.27)$$

$$\nabla_{1,4}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}),$$

$$\nabla_4 \mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = -(\nabla_{1,1}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}))^{-1} \quad (4.28)$$

$$\nabla_{1,5}E(\mathbf{y}^*(\mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}).$$

The Jacobians in (4.27) and (4.28) applied to (4.22) and (4.23), respectively, are referred to as hypergradients in the machine learning literature and are utilized in hyperparameter optimization and meta-learning [48, 102, 105]. Often, approximations of the (inverse) Hessian matrices are made to estimate the hypergradient.

4.3.3 Learning Algorithms

With the complete loss function defined, this section categorizes four principled techniques for learning the neural and symbolic weights of a NeSy-EBM: (1) Modular Learning, (2) Gradient Descent, (3) Bilevel Value-Function Optimization, and (4) Stochastic Policy Optimization. This formulation, framed within the context of NeSy-EBMs, is primarily contributed by my colleague and can be explored in greater detail in [43].

4.3.3.1 Modular Learning

The first and most straightforward NeSy-EBM learning technique is to train and connect the neural and symbolic components as independent modules. This approach has been extensively studied over decades across multiple domains and communities, such as the statistical relational learning community (e.g., [120]).

In a typical modular learning setup, the neural component is trained first using standard methods, such as backpropagation with the Adam optimizer, to minimize a neural loss based on neural labels. Once the neural weights are optimized, they are frozen, and the symbolic component is trained independently using an appropriate method to optimize a value-based or minimizer-based loss. By definition, modular learning algorithms are not trained end-to-end, meaning the neural and symbolic parameters are not jointly optimized to minimize the overall learning loss. Due to this separation, modular approaches may struggle to achieve weight settings that minimize the learning loss as effectively as end-to-end techniques. Nevertheless, modular learning approaches remain appealing and widely used due to their simplicity and broad applicability. There are many well-established and effective modular learning algorithms for both neural and symbolic components. For a recent taxonomy of symbolic weight learning algorithms, see Srinivasan et al. (2021).

4.3.3.2 Gradient Descent

A conceptually simple yet often challenging technique for end-to-end NeSy-EBM training is direct gradient descent. This approach directly utilizes the gradients derived in the previous subsection and from Dickens (2024), applying a gradient-based optimization algorithm to minimize the NeSy-EBM loss with respect to both the neural and symbolic weights.

While backpropagation can be applied to train a wide range of NeSy-EBMs, it is not universally applicable. Scenarios where backpropagation is computationally inexpensive and possible often involve NeSy-EBMs trained with value-based losses. Additionally, for certain subclasses of NeSy-EBMs, gradients of energy minimizers may also exist and be efficient to compute. For instance, when the energy minimizer can be determined through

a simple closed-form expression, such as in cases where inference involves an unconstrained strongly convex quadratic program or a finite computation graph.

However, as discussed in Section 4.3.2, the existence of learning loss gradients for fully expressive NeSy-EBMs depends on specific conditions. Moreover, computing these gradients often requires expensive second-order information about the energy function at the minimizer. Consequently, cheap direct gradient descent is restricted to a relatively small subset of NeSy-EBMs with specialized architectures that enable efficient and principled gradient computation [43].

4.3.3.3 Bilevel Value-Function Optimization

The third class of NeSy-EBM training algorithms leverages bilevel value-function optimization to optimize minimizer-based losses using only first-order gradients. This technique is grounded in the observation that the general definition of NeSy-EBM learning (4.10) is naturally formulated as a bilevel optimization problem. In essence, the NeSy learning objective depends on variable values obtained by solving a lower-level inference problem, which involves symbolic reasoning.

While the details are beyond the scope of this section, my colleague has shown how to reformulate the minimizer-based loss into a bilevel formulation of the NeSy-EBM learning problem, providing a foundation for smooth, first-order gradient-based optimization techniques. Using this formulation, they proposed a NeSy-EBM learning algorithm [43]. The algorithm proceeds by approximately solving instances of a relaxed, smoothed value function in a bound-constrained optimization framework, iteratively refining solutions in a sequence governed by a decreasing parameter ι . This results in a graduated approach, where the problem is addressed through a series of increasingly tighter approximations.

4.3.3.4 Stochastic Policy Optimization

The final approach to NeSy-EBM learning presented here avoids directly computing the energy minimizer’s gradients with respect to the weights by reformulating NeSy learning as a stochastic policy optimization problem. This requires adapting the standard

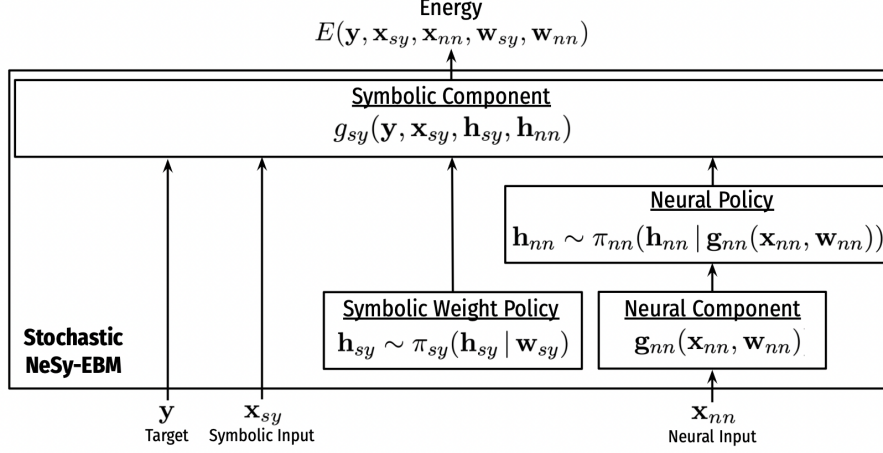


Figure 4.1: A stochastic NeSy-EBM. The symbolic weights and the neural component parameterize stochastic policies. A sample from the policies is drawn to produce arguments of the symbolic component.

NeSy-EBM framework to create a stochastic NeSy-EBM (Figure 4.1). Details on how to perform this reformulation can be found in the work of my colleague [43].

Stochastic policy optimization techniques are broadly applicable for end-to-end training of NeSy-EBMs because they are agnostic to both the neural-symbolic interface and the symbolic inference process. Furthermore, these techniques can be applied across all motivating applications and modeling paradigms.

However, the primary tradeoff of the stochastic policy approach is the high variance in the sample estimates of the policy gradient. This issue is well-documented in the policy optimization literature and becomes more pronounced as the dimensionality of the policy output space increases [123]. As a result, learning with stochastic policy optimization may require significantly more iterations to converge compared to the other techniques presented in this section.

4.4 NeSy Learning Design Principles

Similarly to the inference principles section, the practical implementation of learning in NeSy systems varies significantly depending on the method, task, underlying architecture, and system. While specific NeSy learning algorithms have been developed and tailored to optimize individual NeSy approaches, this section focuses on a set of NeSy design principles that have shown promise across various settings. It is important to emphasize that this discussion centers on the high-level design of learning systems. Detailed aspects of the learning process—such as specific optimization techniques, algorithmic implementations, and related nuances—are beyond the scope of this dissertation. I refer readers to the respective NeSy methods and their specific learning techniques for more granular categorizations and technical details.

This section is organized into three main parts. First, it introduces two data-driven NeSy learning design principles: *distant supervision learning* (Section 4.4.1) and *structure-informed learning* (Section 4.4.2). Distant supervision learning involves labels for the symbolic variables but offers partial or no labels for the neural concepts. In contrast, structure-informed learning operates without labels for the symbolic target variables, while neural components may have full, partial, or no labels. Next, the section discusses an architecture-driven NeSy learning design principle, *learning with constraint loss* (Section 4.4.3), which focuses on a design principle for accelerating learning by incorporating constraints directly into the loss function or have the symbolic perform computation graph execution. Finally, Section 4.4.4 categorizes additional prominent learning design principles, including tasks, pre-training strategies, and post-training approaches.

4.4.1 Distant Supervision Learning

In the first data-driven NeSy learning design pattern, *distant supervision* [87] learning leverages labeled symbolic target variables to indirectly guide the training of the neural component. In the most typical setup for this setting, the neural component does not receive direct supervision from labeled neural data. Instead, the symbolic component provides indirect supervision by propagating information derived from symbolic target

labels associated with a different set of variables.

Formally, in completely unsupervised neural distant supervision learning, the training dataset \mathcal{S} does not include neural labels \mathbf{t}_{nn} :

$$\mathcal{S} := \{(\mathbf{t}_y^1, \mathcal{Z}_{nn}^1, \mathcal{Z}_y^1, \mathbf{x}_{sy}^1, \mathbf{x}_{nn}^1), \dots, (\mathbf{t}_y^P, \mathcal{Z}_{nn}^P, \mathcal{Z}_y^P, \mathbf{x}_{sy}^P, \mathbf{x}_{nn}^P)\}.$$

As a result, the loss function presented in Equation 4.11 simplifies to include only the value-based and minimizer-based losses:

$$\begin{aligned} L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \\ = & L_{Val}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) && \text{Value-Based} \\ & + L_{Min}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) && \text{Minimizer-Based} \end{aligned}$$

Commonly, distant supervision is applied in decomposed task structure settings (Section 4.2.3), where the neural component is trained entirely using labels associated with the symbolic model. For example, in the MNIST-Add scenario introduced in Example 2.1.1, the neural model’s parameters are optimized based on gradient information propagated from the symbolic sum label without direct supervision for the individual digit predictions. This approach is particularly effective when the label associated with the higher-level task (e.g., the sum in MNIST-Add) provides sufficient signal to train the neural model. Further another useful design principle is that once trained, the neural component can be “unplugged” from the NeSy system and deployed independently in other contexts.

With that all said, distant supervision learning is more general than both the decomposed task structure setting and scenarios where there are no labels available to train the neural component. Instead, a hybrid learning strategy can be employed, where the neural model has direct supervision for some examples while relying on distant supervision for the remaining ones. Formally, the training dataset \mathcal{S} includes partially labeled neural data (\mathbf{t}_{nn}) and fully labeled symbolic data (\mathbf{t}_y). This hybrid setup is particularly prevalent in tasks like neural-symbolic semi-supervised distant supervision learning.

4.4.2 Structure-Informed Learning

On the opposite end of the data-driven learning spectrum from distant supervision learning is another common NeSy training design pattern: *structure-informed learning*. In this paradigm, the symbolic target variables lack ground truth labels, while the neural components may have full, partial, or no labels. Instead of relying on explicit supervision, the neural parameters are trained using structurally informed priors or domain-specific knowledge, which act as implicit guidance during the learning process.

Formally, in symbolic-informed learning, the training dataset \mathcal{S} does not include symbolic labels \mathbf{t}_Y :

$$\mathcal{S} := \{(\mathbf{t}_{nn}^1, \mathcal{Z}_{nn}^1, \mathcal{Z}_Y^1, \mathbf{x}_{sy}^1, \mathbf{x}_{nn}^1), \dots, (\mathbf{t}_{nn}^P, \mathcal{Z}_{nn}^P, \mathcal{Z}_Y^P, \mathbf{x}_{sy}^P, \mathbf{x}_{nn}^P)\}.$$

As a result, the loss function presented in Equation 4.11 simplifies to include only the neural-based and value-based losses:

$$\begin{aligned} L^i(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \\ &= L_{NN}(\mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{t}_{nn}^i) && \text{Neural} \\ &\quad + L_{Val}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) && \text{Value-Based} \end{aligned}$$

Furthermore, since labels for symbolic target variables are entirely absent, the value-based learning loss optimal value functions for latent and convolutional are not used and only use the full optimal value function V_Y :

$$V_Y(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) := \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \quad \text{full}$$

Typically a *full-energy loss* is used:

$$L_{Energy}^{full}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) := V_Y(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i),$$

Unlike distant supervision, structure-informed learning is applicable to both unified and decomposed task structures (Section 4.2.3), as the symbolic loss serves as a prior that influences the neural model. For example, in the visual Sudoku puzzle-solving example introduced in the deep symbolic variables section of the modeling paradigms chapter

(Section 3.2.1), the symbolic model lacks supervision on the target variables. In this decomposed task setting, the neural component (digit classifier) is trained solely from the structure of the symbolic model (rules of Sudoku). This approach is particularly effective when domain knowledge or constraints provide a strong inductive bias or are nearly always valid. It is a practical method for training models in scenarios where collecting ground truth labels is challenging or expensive, but the problem can be represented effectively through a set of constraints.

4.4.3 Learning with Constraint Loss

While the previous two design patterns focus on model design from the perspective of data availability, another prominent NeSy learning design pattern emphasizes efficient learning by utilizing the symbolic component as either a constraint loss [138] or a computation graph [14]. This approach, commonly referred to as *learning with constraints*, simplifies the underlying inference optimization process by transforming it into a series of computational steps, analogous to those in a neural network. This design principle is primarily aimed at improving the scalability of training by reducing the computational overhead associated with optimization-based inference. In many cases, it is beneficial to train the neural model on this simplified problem and later transition to a more complex problem involving optimization during inference. For instance, in the MNIST-Add problem (2.1.1), the neural model can be trained using a constraint loss to approximate the addition constraint, while inference employs optimization to ensure that the predictions strictly adhere to the addition rule.

In a learning with constraints design pattern, the neural component is typically designed to predict a subset of symbolic latent variables and symbolic target variables while the remaining variables are resolved trivially through predefined computations. Although this design impacts all learning losses, most systems typically employ a value-based learning loss. Therefore, this section will focus on that aspect in the following discussion.

Formally, the optimal value functions introduce indicator functions $I_{g_{nn}}^{\mathcal{Z}}$ and $I_{g_{nn}}^{\mathcal{Y}}$, which ensure that the latent variables $\hat{\mathbf{z}}$ and target variables $\hat{\mathbf{y}}$ and the neural predictions

$\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ are aligned.¹ Specifically, these indicator functions are defined as:

$$I_{g_{nn}}^{\mathcal{Z}}(\hat{\mathbf{z}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = \begin{cases} 0 & \text{if } \hat{\mathbf{z}} = \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) \\ \infty & \text{otherwise} \end{cases}$$

$$I_{g_{nn}}^{\mathcal{Y}}(\hat{\mathbf{y}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = \begin{cases} 0 & \text{if } \hat{\mathbf{y}} = \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) \\ \infty & \text{otherwise} \end{cases}$$

This then affects the optimal value functions. For the latent optimal value function, the indicator function is incorporated to ensure that the latent variables and neural network outputs are consistent:

$$\begin{aligned} V_{\mathcal{Z}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + I_{\mathcal{T}_{\mathcal{Y}}^i}(\hat{\mathbf{y}}, \mathbf{t}_{\mathcal{Y}}^i) + I_{g_{nn}}^{\mathcal{Z}}(\hat{\mathbf{z}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn})), \\ &= E(\mathbf{t}_{\mathcal{Y}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{w}_{sy}). \end{aligned} \quad (4.29)$$

For the full optimal value function, the indicator function is incorporated to ensure that the target variables and neural network outputs are consistent:

$$\begin{aligned} V_{\mathcal{Y}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i) &= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + I_{g_{nn}}^{\mathcal{Y}}(\hat{\mathbf{y}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn})) \\ &= E(\mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{w}_{sy}). \end{aligned} \quad (4.30)$$

For the convolutional optimal value function, the indicator function is incorporated to ensure that the target variables and neural network outputs are consistent:

$$\begin{aligned} V_{conv}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}_i; \mathbf{y}, \lambda) &= \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}^i, \mathbf{x}_{nn}^i, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + \lambda \cdot d(\hat{\mathbf{y}}, \mathbf{y}) + I_{g_{nn}}^{\mathcal{Y}}(\hat{\mathbf{y}}, \mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn})) \\ &= E(\mathbf{g}_{nn}(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{x}_{sy}^i, \mathbf{w}_{sy}, \lambda). \end{aligned} \quad (4.31)$$

Simplifying the learning problem offers certain benefits but comes with notable trade-offs. On the positive side, this approach often leads to a highly efficient training pipeline, as the inference process becomes computationally trivial. However, this efficiency comes at the cost of reduced model expressivity. The reliance on a computation graph

¹This section assumes that the neural model predicts all latent and target values. Extending this framework to include partial value assignments is straightforward.

limits the ability to represent more complex reasoning, and it increases the likelihood of constraint violations. Additionally, the burden of satisfying constraints is shifted entirely to the neural model, which may struggle in scenarios involving noisy or incomplete data.

4.4.4 Additional Design Guidelines

In addition to the primary learning principles discussed earlier, there are several informal guidelines, tasks, and pipelines that, while not directly modifying the underlying learning loss or algorithms, play a critical role in shaping the design and implementation of NeSy systems.

Neural-Symbolic Tasks: Although NeSy systems are versatile and powerful tools, they are not a universal solution for all problems. Instead, their effectiveness often depends on the nature of the task and the integration of symbolic reasoning with neural learning. Below, I highlight a set of key NeSy tasks that have demonstrated success across a variety of approaches and application domains:

- **Knowledge-Informed Supervised Learning:** Integrates symbolic constraints or priors into a fully supervised neural learning setup to guide the model towards domain-consistent predictions.
- **Semi-Supervised Learning:** Combines limited labeled data with symbolic reasoning or structure to propagate supervision to unlabeled examples, enhancing generalization.
- **Few-/Zero-Shot Learning:** Relies on symbolic reasoning or prior knowledge to train models that generalize effectively from minimal or no labeled examples.
- **Fine-Tuning and Adaptation:** Adapts a pre-trained NeSy model to a new domain or task, often using domain-specific symbolic constraints to guide adaptation.

Pre-training Strategies: While NeSy systems have shown effectiveness in training both neural and symbolic parameters from scratch, there is a growing recognition of the importance of adapting pre-training strategies on the neural side before applying a NeSy loss

[43]. Pre-training can help avoid degenerate solutions and provide a stronger foundation for joint neural-symbolic training. Below are three key pre-training strategies:

- **Supervised Pretraining:** Train the neural model using standard supervised learning techniques on labeled data before introducing a NeSy training pipeline. Supervised pretraining provides the neural component with a robust initial representation, enabling it to better complement the symbolic component during joint training.
- **Self-Supervised Pretraining:** Use self-supervised learning techniques to pretrain the neural model before introducing a NeSy training pipeline. This can range from simple clustering-based approaches to more advanced methods leveraging data augmentations or contrastive learning [24]. Self-supervised pretraining is particularly effective in settings with limited labeled data, providing a robust initialization that can prevent convergence to degenerate solutions or local minima [103]. It is also a practical strategy in scenarios where labeled data is expensive or difficult to obtain.
- **Transfer Learning and Domain Adaptation:** Pretraining large-scale models, such as transformers or foundation models, directly within a NeSy system is often computationally infeasible and may lead to poor generalization. Instead, fine-tuning a pre-trained foundation model or a large computer vision model with symbolic constraints has emerged as a promising technique.

Post-Training Use-Cases: Training a collection of neural and symbolic parameters in a NeSy system does not necessarily imply that the entire model will be used after training. In many cases, NeSy systems are designed to bootstrap the performance of individual components:

- **Symbolic Unplugging:** Train the full NeSy system but utilize only the neural model for inference by “unplugging” the symbolic component. This approach is particularly beneficial in deployment scenarios where computational efficiency is critical. Here, the symbolic component primarily acts as a regularizer during training, ensuring that the neural model adheres to domain-specific constraints.

- **Neural or Symbolic Model Transfer:** Transfer the trained neural or symbolic component to a different domain or task. This strategy leverages the representations or reasoning capabilities learned during training to address new challenges. For example, a neural model pre-trained within a NeSy system can be adapted for tasks requiring similar feature representations, or a symbolic component can be reused to encode domain knowledge in another context.

Chapter 5

Challenges and Pitfalls of NeSy Modeling Paradigmns, Learning, and Reasoning

Throughout the previous two chapters, I have developed and categorized a universal mathematical framework (Section 3.1), a set of modeling paradigms (Section 3.2), and a collection of design principles for NeSy inference and learning (Chapter 4). While these components provide a comprehensive foundation for designing, training, and predicting NeSy approaches, they do not address the practical challenges often encountered during implementation and deployment. In particular, blindly applying NeSy inference and learning techniques without considering the specific requirements and nuances of a task can lead to suboptimal performance or unintended consequences. To address this gap, this chapter organizes and examines common pitfalls in neural-symbolic approaches, categorizing these challenges into three areas: *pitfalls in NeSy modeling paradigms*, *pitfalls in NeSy inference*, and *pitfalls in NeSy learning*. The first section, *pitfalls in NeSy modeling paradigms* (Section 5.1), explores additional modeling paradigms not covered in Section 3.2, along with the inherent pitfalls associated with their design. These paradigms include *unfixed deep symbolic variables (DSVar)* and *deep symbolic operations (DSOp)*. The second section, *pitfalls in NeSy inference* (Section 5.2), discusses three inference-specific pitfalls widely seen in NeSy, including *reasoning shortcuts as unintended optima*, *poor factorization/decomposition*, and *conditional independence in NeSy probabilistic logics*. In the final

section, *pitfalls in NeSy learning* (Section 5.3), I examine prevalent learning-related pitfalls, including *contextual label ambiguity*, *energy-loss degenerate solutions*, and *NeSy soft logic pitfalls*.

5.1 NeSy Modeling Paradigm Pitfalls

In Chapter 3, I introduced three modeling paradigms for neural-symbolic integration that are widely studied across the NeSy community. While these paradigms are robust and versatile, they do not encompass every possible NeSy-EBM design. In fact, combinations of these paradigms offer significant potential and represent an intriguing avenue for future work. However, not all paradigms are free from inherent challenges. This section introduces two additional modeling paradigms—*Unfixed Deep Symbolic Variables* (Section 5.1.1) and *Deep Symbolic Operations* (Section 5.1.2)—that come with inherent pitfalls, particularly in the contexts of inference and learning. It is important to note that while these paradigms present inherent pitfalls, it does not imply that these paradigms are invalid or without merit. Rather, they highlight specific areas where additional considerations are required to overcome these limitations.

5.1.1 Unfixed Deep Symbolic Variables

The *unfixed deep symbolic variables* (Unfixed-DSVar) paradigm is a variant of the *deep symbolic variable* modeling paradigm (Section 3.2.1) in which the neural predictions do not directly control the NeSy-EBM through an indicator function. In an Unfixed-DSVar model, the neural predictions serve as an initialization for the target and latent variables, providing a reasonable starting point for the optimization process. However, once optimization begins, the neural predictions no longer influence the prediction. In this sense, the neural model “seeds” the target and latent variables, but the solution is derived purely through symbolic reasoning. Formally, this modeling paradigm is defined as:

Definition 6. In the *unfixed deep symbolic variables* (Unfixed-DSVar) modeling paradigm, the symbolic potential set is defined as a singleton $\Psi = \{\psi\}$ with a trivial index set

$\mathbf{J}_\Psi = \{1\}$, such that $\Psi_1 = \psi$. Further, the neural prediction is included as the starting value of the symbolic target variables, so that $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy} \times \mathbb{R}^{d_{nn}}$. Unlike in fixed DSVar models, these initial neural predictions do not determine the final values of the target variables \mathbf{y} .

The symbolic component, governed by the symbolic potential, is thus:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \psi([\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})], \mathbf{w}_{sy}),$$

where $[\cdot]$ denotes concatenation.

In the *Unfixed-DSVar* setting, the gradient can no longer be directly computed with respect to the neural model’s predictions.¹ Instead, an additional mechanism is required to calculate the dissatisfaction of the optimal values assigned after symbolic optimization and the original neural predictions. Effectively, the symbolic model produces a label that the neural model aims to align with a separate loss function:

$$d(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \hat{\mathbf{y}})$$

where $d(\cdot, \cdot)$ is a loss function, and $\hat{\mathbf{y}}$ represents the optimal values for target and latent variables:

$$\hat{\mathbf{y}} = \arg \min_{\hat{\mathbf{y}} \in \mathcal{Y}} E(\hat{\mathbf{y}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$$

While this modeling paradigm is less commonly studied, it does offer unique utility in certain scenarios. Specifically, it is useful when greater control over the loss function within the neural model is desired, as the incorporation of the distance function $d(\cdot, \cdot)$ allows for flexibility in defining the neural loss with respect to the symbolic predictions. That said, without careful design of both the distance function and the symbolic model, this approach is prone to significant pitfalls.

Inherent Pitfall The Unfixed-DSVar modeling paradigm introduces the risk that symbolic reasoning may disregard neural predictions altogether if they conflict with the constraints. This can lead to symbolic solutions that satisfy constraints independently of the

¹Technically, a gradient can be propagated through this variable; however, it will align with the optimal predictions of the symbolic model rather than the original neural predictions.

neural model’s input, potentially undermining the neural component’s intended role. For instance, in the visual sudoku puzzle classification scenario (Figure 3.2), the neural model’s initial predictions for each digit will seed the target variables in the symbolic component, however, symbolic optimization could then completely ignore this prediction. To illustrate, consider that any valid solution has the same energy value of zero since there is no limitation on assigning the symbolic solution close to the neural prediction. In this scenario, optimization can be trivially accomplished by assigning every assignment of neural variables to a default solution.

5.1.2 Deep Symbolic Operations

The *deep symbolic operations* (DSOp) paradigm is a specialized class of the *deep symbolic potential* modeling paradigm (Section 3.2.3) in which the neural component predicts the symbolic operations such as logical conjunctions, disjunctions, and negations. In a DSOp model, the neural model is a generative model that samples symbolic operations from a set for each grounded potential. This modeling paradigm defines NeSy-EBM’s that can represent the *sampling neural for symbolic* architectural axiom Section 2.3.1.

Definition 7. *In the **deep symbolic operations** (DSOp) modeling paradigm, the symbolic potential set Ψ is the set of all potential functions that can be created from all combinations of operations in the original provided symbolic model. Given k operations and m values an operation can take (e.g., $\wedge \vee \neg$), Ψ is indexed by the output of the neural component, i.e., $\mathbf{J}_\Psi = \text{Range}(\mathbf{g}_{nn}) = \{0, 1, \dots, k^m\}$ and $\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}$ is the potential function indexed by the neural prediction. The variable and parameter domains of the sampled symbolic potential are $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$, and $\text{Params}_\psi = \mathcal{W}_{sy}$, respectively. The symbolic component expressed via the symbolic potential is:*

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}([\mathbf{y}, \mathbf{x}_{sy}], \mathbf{w}_{sy}). \quad \square$$

Inherent Pitfalls Unlike the DSPot NeSy-EBM modeling paradigm, the DSOp paradigm enforces a restriction that the random variables and structure of the problem remain fixed. While this constraint ensures stability and drastically reduces the search space inherent to

the problem, it imposes a significant limitation on the types of tasks the DSOp paradigm can effectively represent. One of the key advantages of the DSPot paradigm is its applicability to truly open-ended tasks, a flexibility that the DSOp paradigm inherently lacks. Furthermore, altering the operational values in a DSOp system can disrupt the specialized optimization processes employed by many existing systems. For instance, the most prominent NeSy logic systems rely heavily on Horn clauses for their efficiency and structure. Allowing a neural model to modify operations within these systems would compromise their highly optimized inference and learning techniques.

5.2 NeSy Inference Pitfalls

In the previous section, I examined pitfalls associated with modeling paradigms; this section shifts the focus to pitfalls that arise during inference. These challenges are discussed before addressing learning-related issues, as inference often serves as a subprocess within learning pipelines, and its shortcomings can exacerbate learning-related problems. Specifically, this section explores problems such as *reasoning shortcuts as unintended optima* (Section 5.2.1), *poor factorization and decomposition* (Section 5.2.2), and *conditional independence in NeSy probabilistic logics* (Section 5.2.3). Further, within each subsection I describe each of these pitfalls through NeSy-EBMs, highlighting their causes and implications for model performance, scalability, and generalization.

5.2.1 Reasoning Shortcuts as Unintended Optima

Reasoning shortcuts in NeSy approaches, originally introduced by Marconato et al. (2023), occur when models attain high accuracy by leveraging concepts with unintended semantics. In this section, I introduce reasoning shortcuts as presented by Marconato et al. (2023) and then generalize it within the NeSy-EBM framework. I refer the reader to the original work for further technical details and proofs.

Reasoning Shortcuts in NeSy Predictors To introduce reasoning shortcuts, consider the concept of *NeSy predictors*. A NeSy predictor infers target variables $\mathbf{y} \in \mathcal{Y}$ by reasoning

over a set of k discrete concepts $\mathbf{c} \in \mathcal{C}$, which are extracted from a sub-symbolic model based on input data $\mathbf{x} \in \mathcal{X}$. The objective of reasoning over these concepts is to encourage the neural model’s predictions to satisfy a logical formula K , represented as $\mathbf{c} \models K$.

Example 5.2.1. *In the MNIST-Addition setting, given a pair of MNIST digit images, for instance, $\mathbf{x} = (\mathbf{3}, \mathbf{5})$, the neural model infers concepts $\mathbf{c} = (c_1, c_2)$ representing the digit classes, which are then combined to predict their sum $y = 8$.*

In this setting, a reasoning shortcut arises when the NeSy predictor acquires concepts that satisfy the logical formula K but do not accurately represent the true underlying concepts. Formally, define reasoning shortcuts as follows:

Definition 8. *A reasoning shortcut is a distribution $p_\theta(\mathbf{c}|\mathbf{x})$ that achieves maximal log-likelihood on the training set \mathcal{S} but does not match the ground-truth concept distribution $p^*(\mathcal{T}_{\mathbf{c}}|\mathbf{x})$:*

$$L(p_\theta, \mathcal{S}, K) = \max_{\theta' \in \Theta} L(p_{\theta'}, \mathcal{S}, K) \quad \text{and} \quad p_\theta(\mathbf{c}|\mathbf{x}) \neq p^*(\mathcal{T}_{\mathbf{c}}|\mathbf{x}), \quad (5.1)$$

where $L(p_\theta, \mathcal{S}, K)$ denotes the log-likelihood of the model given the training data \mathcal{S} and the logical formula K .

For instance, in the MNIST-Addition example, a reasoning shortcut might assign $\mathbf{3} = 0$ and $\mathbf{5} = 8$. Here, the sum $y = 8$ is correctly predicted, but the concepts do not reflect the correct digit classes, misrepresenting the true semantic meaning.

Generalizing to NeSy-EBMs Reasoning shortcuts can trivially be generalized to the NeSy-EBM framework, where energy functions govern the compatibility of predictions with observed data and constraints. Let \mathbf{x}_{sy} denote the observed symbolic inputs, \mathbf{y} the target variables, and \mathbf{w}_{sy} symbolic parameters. Define a neural network \mathbf{g}_{nn} with parameters $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ and inputs $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$ such that:

$$\mathbf{g}_{nn} : \mathcal{W}_{nn} \times \mathcal{X}_{nn} \rightarrow [0, 1]^n,$$

where n is the number of desired concepts \mathbf{c} . The probability distribution of the concepts is defined by the neural network outputs:

$$p_{\mathbf{w}_{nn}}(\mathbf{c}|\mathbf{x}_{nn}) = \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}).$$

Let $C(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy})$ represent the set of constraints that encode the logical formula K . Now define reasoning shortcuts in the NeSy-EBM context as follows:

Definition 9. *A reasoning shortcut in NeSy-EBMs is a distribution $p_{\mathbf{w}_{nn}}(\mathbf{c}|\mathbf{x}_{nn})$ that maximizes the NeSy-EBM learning loss function over the training data \mathcal{S} , subject to the constraints $C(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy})$, but deviates from the true concept distribution $p^*(\mathcal{T}_{\mathbf{c}}|\mathbf{x}_{nn})$:*

$$\mathcal{L}(E(\cdot, \cdot, \cdot, \hat{\mathbf{w}}_{sy}, \hat{\mathbf{w}}_{nn}), \mathcal{S}) = \max_{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}} \mathcal{L}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}) \text{ and } p_{\mathbf{w}_{nn}}(\mathbf{c}|\mathbf{x}_{nn}) \neq p^*(\mathcal{T}_{\mathbf{c}}|\mathbf{x}_{nn}).$$

Reasoning Shortcuts in Practice This pitfall is broad and encompasses many practical issues encountered when training these methods. Consider the following common reasoning shortcut scenarios:

- **Minimal Constraint Satisfaction Bias:** This shortcut occurs when the model satisfies the constraints as simply as possible. The NeSy predictor may adopt default labels or limited concept representations that satisfy the constraints $C(\cdot)$ without genuinely representing the diversity of concepts. For example, if the label “traffic lights” appears in the fewest constraints, the model may assign all concepts as “traffic lights” to meet the logical requirements with minimal complexity.
- **Misaligned Concepts:** In this shortcut, the model learns concepts that satisfy the constraints $C(\cdot)$ but fail to align with the correct underlying concepts. This can lead to multiple valid solutions that do not necessarily reflect the true concept distribution. For example, in the MNIST-Addition scenario, predicting $\mathbf{3} + \mathbf{5} = 8$ could be satisfied by multiple pairs such as $1 + 7 = 8$, $3 + 5 = 8$, etc.
- **Overfitting to Symbolic Constraints:** This shortcut arises when the model overfits to the symbolic constraints, focusing solely on satisfying the constraints $C(\cdot)$ in the training data without learning the underlying task-specific concepts. In this scenario, the model may achieve low training error by perfectly adhering to the symbolic constraints, but it will generalize poorly, resulting in high test error. Essentially, the

neural model becomes a “yes man” to the symbolic component during training, prioritizing constraint satisfaction over the true desired concepts. For instance, consider a NeSy model trained in an unsupervised *structure-informed learning* setting (Section 4.4.2) with a set of prior constraints that are soft (i.e., not always correct). If the neural model is trained solely to minimize violations of these constraints, it may fail to capture the true relationships in the data. This overfitting can result in a model that aligns well with the symbolic priors during training but performs poorly in real-world scenarios where the priors are imperfect or incomplete.

5.2.2 Poor Factorization/Decomposition

In directed graphical model machine learning theory, *factorization*, or *decomposition*, involves breaking down a joint probability distribution into a structured representation that captures the conditional dependencies among variables [69]. The chosen factorization plays a critical role in determining the efficiency and accuracy of computations. However, identifying the optimal structure for a given problem can be challenging, especially in complex or computationally intensive scenarios. Building on this concept, *poor factorization*, as framed in this thesis, occurs when the structure of the chosen computation graph (Section 4.2.1) results in an inefficient or suboptimal representation. Specifically, the computation graph must define an order of influence among the observed, target, and latent variables during its construction. Due to the inherent complexity or the computational cost of solving the optimization problem directly, the resulting factorization may fail to capture the problem’s true dependencies effectively. To illustrate this concept, consider the following example to illustrate poor factorization in a NeSy problem:

Example 5.2.2. *Consider the task of determining the order of objects in a line, where the goal is to rank a set of candidates, $\{a, b, c\}$, based on predicted placement costs. Rather than solving an exhaustive optimization problem, which may be computationally prohibitive, a computation graph-based approach might simplify this by selecting one specific ordering structure to approximate a solution. For instance, the ordering could be factored into one*

of six possible configurations:

$$a \rightarrow b \rightarrow c, \quad a \rightarrow c \rightarrow b, \quad b \rightarrow a \rightarrow c, \quad b \rightarrow c \rightarrow a, \quad c \rightarrow a \rightarrow b, \quad c \rightarrow b \rightarrow a.$$

Suppose a neural model predicts placement costs for each object in each position as follows:

$$a = \{\text{first} = 0.5, \text{second} = 0.1, \text{third} = 0.4\},$$

$$b = \{\text{first} = 0.3, \text{second} = 0.2, \text{third} = 0.7\},$$

$$c = \{\text{first} = 0.1, \text{second} = 0.0, \text{third} = 0.9\}.$$

Given an ordering for the computation graph, such as $a \rightarrow b \rightarrow c$, a greedy solution will select positions in the order $[b, a, c]$, yielding a total cost of $0.1 + 0.3 + 0.9 = 1.3$. However, this is not the optimal cost. For instance, a computation graph that predicts random variables $c \rightarrow b \rightarrow a$ will yield an order of $[b, c, a]$, with a total cost of $0.0 + 0.3 + 0.4 = 0.7$.

Formally, let \mathbf{x}_{sy} denote the observed symbolic variables, \mathbf{y} the target variables, and \mathbf{w}_{sy} the symbolic parameters. Define the neural network component \mathbf{g}_{nn} with parameters $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ and neural inputs $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$:

$$\mathbf{g}_{nn} : \mathcal{W}_{nn} \times \mathcal{X}_{nn} \rightarrow [0, 1]^n,$$

where n is the number of desired concepts \mathbf{c} . Let \mathcal{G} represent the computation graph structure and $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ as the energy function. Define *poor factorization* as follows:

Definition 10. A poor factorization in NeSy-EBMs occurs when the chosen computation graph structure \mathcal{G} leads to a solution that maximizes the NeSy-EBM learning loss function over the training data \mathcal{S} but results in a suboptimal solution for the task. Formally, let $\hat{\mathbf{y}}_{\mathcal{G}}$ denote the optimal solution given factorization \mathcal{G} and target configuration:

$$\hat{\mathbf{y}}_{\mathcal{G}} = \arg \min_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}),$$

where \mathcal{G} is the chosen computation graph factorization. A poor factorization occurs if there exists a different factorization \mathcal{G}' yielding solution $\hat{\mathbf{y}}_{\mathcal{G}'}$ such that:

$$E(\hat{\mathbf{y}}_{\mathcal{G}}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) > E(\hat{\mathbf{y}}_{\mathcal{G}'}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}).$$

In the example above, poor factorization arises if the computation graph factorization \mathcal{G} selects an ordering, such as $a \rightarrow b \rightarrow c$, which does not minimize the true objective function compared to an alternative factorization. Consequently, while the solution is valid, it is suboptimal.

5.2.3 Conditional Independence in NeSy Probabilistic Logics

This section follows the original work presented by Krieken et al. (2024). To simplify parameter estimation and model training, NeSy probabilistic logic (Section 2.3.3) approaches [138, 81, 128, 7] often assume *conditional independence* of the target variables \mathbf{y} given the observed variables \mathbf{x}_{nn} and \mathbf{x}_{sy} . This implies that the probability of a world w (i.e., an assignment of variables) $p_{\theta}(w|x)$ can be simplified to a product of independent probabilities. As a result, rather than estimating the joint probability for each of the 2^n possible worlds, the neural model only needs to predict n probabilities for each variable and then calculate the probability of the world as:

$$p_{\theta}(w|x) := \prod_{i=1}^n p_{\theta}(w_i|x).$$

This assumption reduces computational complexity, speeds up inference, and significantly reduces the number of trainable parameters, as shown in [138, 81, 128, 7].

Inherent Pitfalls While conditional independence simplifies the learning process, it introduces pitfalls that impact both inference quality and optimization:

- **Bias Toward Deterministic Solutions:** The conditional independence assumption biases the model toward deterministic solutions, as the independence limits the model’s ability to represent joint uncertainty across variables. For example, consider a setup with two binary variables r and g , representing a red and green light, with

a constraint $\phi = \neg r \vee \neg g$ that prohibits both lights from being on simultaneously. Under conditional independence, the probability $p_{\theta}(\phi = 1)$ can be maximized by enforcing $p_{\theta}(r, g) = 0$, leading to a deterministic outcome where either r or g must be off, even if the real-world distribution allows both to remain uncertain.

- **Non-Convex and Disconnected Minima in Semantic Loss:** The independence assumption creates a non-convex and often disconnected landscape in the semantic loss function, leading to isolated minima. For example, the semantic loss minima can form disconnected clusters in simple problems like XOR where dependent variables have two disconnected solution spaces.

5.3 NeSy Learning Pitfalls

In the previous section, I outlined a set of practical challenges commonly encountered during inference across various NeSy approaches. In this section, I shift the focus to learning-specific pitfalls. It is important to note that, in many NeSy approaches, inference operates as a subprocess of learning. Consequently, the inference pitfalls discussed earlier can compound or confound learning challenges, leading to unintended or suboptimal outcomes. While these issues could also be categorized as learning pitfalls, this section focuses specifically on challenges that arise directly from the learning process itself. This section examines three key pitfalls that impact a wide range of NeSy approaches: *contextual label ambiguity* (Section 5.3.1), *energy loss degenerate solutions* (Section 5.3.2), and *NeSy soft logic pitfalls* (Section 5.3.3).

5.3.1 Contextual Label Ambiguity

To begin, I introduce a prevalent data-centric learning pitfall often encountered by newcomers to the NeSy field: *contextual label ambiguity*. At its core, this pitfall manifests as a form of training label noise, where contextually ambiguous labels arise because the neural inputs alone cannot distinguish the underlying target labels. In the context of NeSy methods, this ambiguity occurs when the symbolic structure provides crucial contex-

tual information needed to disambiguate labels, but the neural model lacks access to this knowledge during input processing. To better illustrate this pitfall, consider the following example:

Example 5.3.1. *Consider the visual Sudoku puzzle-solving task introduced in Section 3.2.1. The goal is to train a model capable of solving partially filled visual Sudoku boards populated with MNIST images, ensuring that no digits are repeated in any row, column, or square. In this scenario, the NeSy method operates within a decomposed task structure (Section 4.2.3), where the neural model predicts individual digits, and the symbolic model enforces Sudoku rules to solve the puzzle. The learning process assumes no explicit labels for the MNIST digits; instead, the model relies on a structure-informed learning setting (4.4.2). In this approach, the system must learn solely from the Sudoku constraints, leveraging the structure of the problem to guide its neural digit predictions.*

In this setting, contextual label ambiguity arises if the neural model is naively trained using every cell as independent input data, including blank squares. This will result in a neural component that is unable to distinguish the labels of the blank squares, which in turn hinders the training performance. To illustrate this pitfall, consider the following 4×4 Sudoku board, where some cells are blank while others are populated with MNIST images:

0	5		3
8	3		0
3			
		3	

In this scenario, contextual label ambiguity will occur if the model uses the following dataset:

$$x_{nn} = \left\{ \begin{array}{l} \boxed{0}, \boxed{5}, \boxed{}, \boxed{3}, \boxed{8}, \boxed{3}, \boxed{}, \boxed{0}, \\ \boxed{3}, \boxed{}, \boxed{}, \boxed{}, \boxed{}, \boxed{}, \boxed{3}, \boxed{} \end{array} \right\}$$

Formal Definition in NeSy-EBM Framework Let \mathbf{x}_{sy} represent the observed symbolic variables (e.g., partial board configurations), \mathbf{y} the target variables (e.g., complete

Sudoku solution), and \mathbf{w}_{sy} the symbolic parameters. Let \mathbf{g}_{nn} denote the neural network component with parameters $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ and neural inputs $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$:

$$\mathbf{g}_{nn} : \mathcal{W}_{nn} \times \mathcal{X}_{nn} \rightarrow [0, 1]^n,$$

where n is the number of desired concepts \mathbf{c} (e.g., potential digit classifications for each cell). The distribution of the concepts is defined by the neural network outputs:

$$d_{\mathbf{w}_{nn}}(\mathbf{c}|\mathbf{x}_{nn}) = \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}).$$

Let $C(\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{x}_{sy}, \mathbf{w}_{sy})$ represent the global constraints necessary for accurate predictions in the NeSy-EBM. Define *contextual label ambiguity* as follows:

Definition 11. *Contextual label ambiguity in NeSy-EBMs occurs when, for neural inputs \mathbf{x}_{nn}^A and \mathbf{x}_{nn}^B , the optimal configuration of target variables \mathbf{y} under the energy function E yields distinct values $\hat{\mathbf{y}}^A$ and $\hat{\mathbf{y}}^B$ that cannot be differentiated based on local neural inputs:*

$$\arg \min_{\mathbf{y}} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \hat{\mathbf{y}}$$

with $\hat{\mathbf{y}}^A \in \hat{\mathbf{y}} \neq \hat{\mathbf{y}}^B \in \hat{\mathbf{y}}$ and the following always true:

$$d_{\mathbf{w}_{nn}}(\mathbf{c}^A|\mathbf{x}_{nn}^A) = d_{\mathbf{w}_{nn}}(\mathbf{c}^B|\mathbf{x}_{nn}^B).$$

5.3.2 Energy Loss Degenerate Solutions

While the previous section highlighted a data-centric learning pitfall, this section discusses degenerate solutions achieved during the learning process. Specifically, this section focuses on energy-loss degenerate solutions that can occur during learning. Within the NeSy-EBM framework, *degenerate solutions* in energy-based learning occurs when the symbolic parameters lead to an energy configuration that fails to distinguish between different target variable assignments effectively. This results in what is referred to as a *collapsed energy function*. A common form of degenerate solution arises when the symbolic parameters are set to extreme values, such as all zeros or infinities. In such cases, the energy landscape becomes uniform across all possible target variable configurations, effectively eliminating

the ability of the system to differentiate between valid and invalid assignments. Consequently, inference in this scenario becomes trivial and uninformative, as every assignment achieves the same minimal energy. To illustrate, consider the energy loss defined in the previous section:

$$L_{\text{energy}}(E(\cdot, \cdot, \cdot, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}) = V_{\mathcal{Z}},$$

where the optimal value function $V_{\mathcal{Z}}$ is defined as:

$$V_{\mathcal{Z}}(\mathbf{w}_{sy}, \mathbf{w}_{nn}, \mathcal{S}) := \min_{\mathbf{y} \in \mathcal{Y}} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) + I_{\mathcal{T}_{\mathbf{y}}}(\mathbf{y}, \mathbf{t}),$$

where $I_{\mathcal{T}_{\mathbf{y}}}(\mathbf{y}, \mathbf{t})$ is an indicator function enforcing constraints on the target configurations.

A collapsed energy function can emerge in many scenarios, for instance when the symbolic parameters interact multiplicatively with strictly positive symbolic potentials, $\Phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) > 0$:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \mathbf{w}_{sy}^T \Phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}).$$

Since $\Phi(\cdot)$ is strictly positive across all configurations, the energy can be minimized by reducing \mathbf{w}_{sy} to its smallest feasible values:

- If $\mathbf{w}_{sy} \in \mathbb{R}_+^r$, the degenerate solution is $\mathbf{w}_{sy} = \mathbf{0}$,
- If $\mathbf{w}_{sy} \in \mathbb{R}^r$, the degenerate solution occurs at $\mathbf{w}_{sy} \rightarrow -\infty$.

In both cases, the degenerate solution results in a collapsed energy function, where all configurations of \mathbf{y} are assigned the same minimal energy. Consequently, the model loses predictive power, as it cannot differentiate between high and low-energy states, nor infer meaningful solutions.

Mitigation Strategies To prevent collapsed energy degenerate solutions, several mitigation strategies can be employed. For instance, if the symbolic parameters and the symbolic potentials are strictly positive, then some mitigation strategies to avoid the collapsed energy are:

- **Simplex Constraint:** Introducing a simplex constraint on \mathbf{w}_{sy} , i.e., $\mathbf{w}_{sy} \in \Delta^r := \{\mathbf{w} \in \mathbb{R}_+^r \mid \|\mathbf{w}\|_1 = 1\}$, prevents zero-valued solutions, as $\mathbf{0}$ is not included in the simplex. This constraint ensures that at least one symbolic parameter is always active, preserving the ability of the model to differentiate between configurations of \mathbf{y} .
- **Negative Logarithmic Regularization:** Adding a negative logarithmic penalty to the objective discourages symbolic parameters from approaching zero by assigning high energy to solutions with small symbolic parameters. The regularized learning objective becomes:

$$\mathcal{L}_{\text{energy}}(\mathbf{w}_{nn}, \mathbf{w}_{sy}, \mathcal{S}) - \sum_{i=1}^r \log(\mathbf{w}_{sy}[i]),$$

which penalizes zero weights, encouraging a distribution over \mathbf{w}_{sy} that leverages multiple symbolic components and enhances model robustness by ensuring that multiple potential functions contribute to the energy landscape.

5.3.3 NeSy Soft Logic Pitfalls

Soft logic approaches in NeSy methods models aim to provide differentiable alternatives to traditional binary logic, enabling gradients to flow through soft logical operations. While this flexibility is critical for integrating symbolic reasoning with neural models, soft logic introduces unique challenges that can hinder performance and learning stability. Below, I detail two key pitfalls associated with NeSy soft logic I have commonly encountered during learning:

- **Non-Binary Satisfaction of Values:** Unlike traditional binary logic, which enforces strict satisfaction or violation of constraints with values that are either 0 or 1, soft logic permits intermediate values to fully satisfy constraints. For example, when observing the disjunction for soft logic relaxation of Łukasiewicz logic (Section 2.1), the satisfaction of a rule such as $A \vee B$ is defined as:

$$S(A \vee B) = \min(1, A + B),$$

where S represents the satisfaction score. This relaxation introduces a range of input values that fully satisfy the constraint, for example:

1. If $A = 0.5$ and $B = 0.5$, the satisfaction score is:

$$S(A \vee B) = \min(1, 0.5 + 0.5) = \min(1, 1) = 1.$$

2. If $A = 0.8$ and $B = 0.5$, the satisfaction score is:

$$S(A \vee B) = \min(1, 0.8 + 0.5) = \min(1, 1.3) = 1.$$

In both cases, the satisfaction score achieves its maximum value of 1, even though the individual inputs are not at their extreme values of 1. This creates a region of full satisfaction where the gradient of the satisfaction function with respect to A and B is zero. Consequently, during gradient-based learning, once the model reaches this region, it ceases to receive meaningful updates for these variables, halting further improvements.

This phenomenon allows the model to converge to representations that satisfy all constraints without pushing variable values to their extremes. While this behavior can be beneficial in some contexts, it can also lead to suboptimal solutions in tasks requiring high confidence or strict logical adherence. Similar observations have been made by van Krieken et al. (2023) and Evans and Grefenstette (2018), among others, who highlight that soft logic relaxations, while enabling differentiability, can undermine interpretability and robustness by tolerating ambiguous intermediate solutions.

- **Equal Gradient Across Variables Leading to Averaged Predictions:** In categorical problems where multiple classes satisfy a given constraint, gradients passed back to the model can become equally distributed among those classes. This results in predictions that remain evenly weighted across the valid classes rather than favoring any one solution. This issue arises because the model either lacks additional information to differentiate between equally valid solutions or does not employ mechanisms (e.g., sampling or structured decision-making) to resolve the ambiguity.

For example, in an MNIST addition task (Example 2.1.1), suppose the sum of two digits is constrained to be 2. There are three valid assignments that satisfy this constraint: $(0+2)$, $(1+1)$, and $(2+0)$. When computing gradients, the model receives positive gradients for the digits $\{0, 1, 2\}$ and negative gradients for all other digits $\{3, 4, 5, \dots, 9\}$. This feedback effectively informs the model that all three assignments are equally valid but provides no mechanism to prioritize one over the others.

As a result, the predictions for $\{0, 1, 2\}$ remain equally weighted, leaving the model unable to converge to a stronger, more confident assignment for any of these classes individually. This issue becomes further compounded when combined with the non-binary satisfaction of values inherent in soft logic systems, as the model may never develop sufficient confidence in its predictions. Without additional data, auxiliary constraints, or structural modifications to guide learning, the model risks producing overly ambiguous or diluted predictions, particularly in tasks requiring high certainty or fine-grained distinctions.

Part IV

A General and Principled Neural-Symbolic Implementation

Chapter 6

Deep Hinge-Loss Markov Random Fields and Neural Probabilistic Soft Logic

Throughout the previous four chapters, I have established the foundational components necessary for developing a principled neural-symbolic approach. These components include a set of architectural axioms (Chapter 2), a universal language and modeling paradigms to formalize these axioms (Chapter 3), and a comprehensive set of inference and learning design principles optimization and reasoning (Chapter 4). Together, these components provide a cohesive theoretical foundation for designing robust, scalable, and principled NeSy systems.

With this foundational theory in place, I identify a significant gap in the existing landscape of NeSy systems: the lack of a framework that operates on fuzzy logic semantics, supports multiple architectural axioms, and prioritizes complex optimization-based inference over computation graph execution (Section 4.2.1). This chapter addresses this gap by introducing Neural Probabilistic Soft Logic (NeuPSL), a novel NeSy approach that implements a collection of the design principles, modeling paradigms, and architectural axioms developed throughout this dissertation. This chapter is structured as follows: first, I extend hinge-loss Markov random fields to incorporate deep components (Section 6.1) and discuss inference and learning (Section 6.2). Then, I introduce NeuPSL syntax and semantics (Section 6.3), followed by an explanation of how NeuPSL can represent all NeSy-

EBM modeling paradigms (Section 6.4). Finally, I provide a brief overview of the NeuPSL system design (Section 6.5).

6.1 Deep Hinge-Loss Markov Random Fields

Much of my contribution builds extensively on Probabilistic Soft Logic (PSL) [11], a highly scalable probabilistic programming language that simplifies the instantiation and application of a specialized class of Markov random fields known as *hinge-loss Markov random fields* (HL-MRFs). In this section, I begin with a detailed formalization of HL-MRFs [11] in which I discuss their hinge-loss energy function and the associated probability density function that facilitates inference. Building on this foundation, I will then introduce *Deep Hinge-Loss Markov Random Fields* (Deep HL-MRFs), which extends the standard HL-MRF definition to allow random variables and parameters to be defined using a neural network, enabling more expressive and flexible models for handling complex and high-dimensional data.

6.1.1 Hinge-Loss Markov Random Fields

Hinge-loss Markov random fields (HL-MRFs) are a family of convex probabilistic models designed for scalable and accurate reasoning over both discrete and continuous data. These models utilize *hinge-loss potentials* to capture constraints and dependencies among variables, enabling flexible probabilistic inference across complex relational structures. In the following, I will formally define HL-MRFs as presented in Bach et al. (2017) by introducing their hinge-loss energy function and the associated probability density function that governs their behavior.

Definition 12 (Hinge-Loss Markov Random Field). *Let $\mathbf{y} = (y_1, \dots, y_n)$ be a vector of n continuous random variables, and $\mathbf{x} = (x_1, \dots, x_m)$ be a vector of m observed continuous variables, with joint domain $\mathcal{D} = [0, 1]^{n+m}$.*

*Define $\phi = (\phi_1, \dots, \phi_m)$ as a vector of m continuous **hinge-loss potentials**, each of the form*

$$\phi_j(\mathbf{y}, \mathbf{x}) = (\max\{\ell_j(\mathbf{y}, \mathbf{x}), 0\})^{p_j}, \quad (6.1)$$

where ℓ_j is a linear function over \mathbf{y} and \mathbf{x} , and $p_j \in \{1, 2\}$.

Let $\mathbf{c} = (c_1, \dots, c_r)$ represent a vector of r linear constraint functions associated with index sets for equality constraints E and inequality constraints I . Define the **feasible set** as:

$$\tilde{\mathcal{D}} = \left\{ (\mathbf{y}, \mathbf{x}) \in \mathcal{D} \left| \begin{array}{ll} c_k(\mathbf{y}, \mathbf{x}) = 0, & \forall k \in E, \\ c_k(\mathbf{y}, \mathbf{x}) \leq 0, & \forall k \in I \end{array} \right. \right\}. \quad (6.2)$$

For a vector of non-negative weights $\mathbf{w} = (w_1, \dots, w_m)$, define the **hinge-loss energy function** as:

$$E(\mathbf{y}, \mathbf{x}, \mathbf{w}) = \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}). \quad (6.3)$$

An **hinge-loss Markov random field** P over variables \mathbf{y} conditioned on \mathbf{x} is a probability density defined as follows:

$$P(\mathbf{y}|\mathbf{x}; \mathbf{w}) = \begin{cases} \frac{1}{Z(\mathbf{w}, \mathbf{x})} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{w})) & \text{if } (\mathbf{y}, \mathbf{x}) \in \tilde{\mathcal{D}}, \\ 0 & \text{otherwise,} \end{cases} \quad (6.4)$$

where the partition function $Z(\mathbf{w}, \mathbf{x})$ is given by:

$$Z(\mathbf{w}, \mathbf{x}) = \int_{\mathbf{y} | (\mathbf{y}, \mathbf{x}) \in \tilde{\mathcal{D}}} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{w})) d\mathbf{y}. \quad (6.5)$$

A key advantage of hinge-loss Markov random fields is their reliance on convex linear functions, which yield a convex objective function. This convexity allows for efficient and rapid inference, making HL-MRFs highly scalable for large and complex datasets.

6.1.2 Deep Hinge-Loss Markov Random Fields

While HL-MRFs provide a powerful framework for modeling complex relational dependencies with convexity properties that facilitate efficient inference, they are limited by the assumption that the potential functions and constraints are defined solely through linear combinations of observed, latent, and target variables defined with a database. To enhance the expressive capacity of HL-MRFs, I propose *Deep Hinge-Loss Markov Random Fields* (Deep HL-MRFs), which extend the traditional HL-MRF framework by allowing

random variables and parameters to be specified from a neural network in a way that allows for differentiable learning.

Definition 13 (Deep Hinge-Loss Markov Random Field). *Let $\mathbf{y} = (y_1, \dots, y_{n_y})$ be a vector of n_y continuous random variables, and $\mathbf{x}_{sy} = (x_{sy,1}, \dots, x_{sy,n_{sy}})$ be a vector of n_{sy} continuous observed symbolic variables, both within the domain $\mathcal{D} = [0, 1]^{n_y+n_{sy}}$.*

Let $\mathbf{x}_{nn} = (x_{nn,1}, \dots, x_{nn,n_{nn}})$ be a vector of n_{nn} neural input features, and $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ represent a set of neural networks parameterized by \mathbf{w}_{nn} . This set of neural networks can represent random variables or parameters:

$$\begin{aligned} \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) : \mathcal{X}_{nn} \times \mathcal{W}_{nn} &\rightarrow [0, 1]^{n_{nn}^{RV}}, \\ \mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) : \mathcal{X}_{nn} \times \mathcal{W}_{nn} &\rightarrow [0, \infty)^{n_{nn}^{Param}}, \end{aligned}$$

where \mathbf{g}_{nn}^{RV} outputs n_{nn}^{RV} continuous random variables within the interval $[0, 1]$, and \mathbf{g}_{nn}^{Param} provides n_{nn}^{Param} non-negative parameters for the potentials.

Define $\phi = (\phi_1, \dots, \phi_m)$ as a vector of m continuous **deep hinge-loss potentials**, each of the form

$$\phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = (\max\{\ell_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), 0\})^{p_j}, \quad (6.6)$$

where ℓ_j is a linear function over \mathbf{y} , \mathbf{x}_{sy} , and $\mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, with $p_j \in \{1, 2\}$. Let $\mathbf{c} = (c_1, \dots, c_r)$ represent a vector of r linear constraint functions, each dependent on \mathbf{y} , \mathbf{x}_{sy} , and $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, associated with index sets for equality constraints E and inequality constraints I . Define the **feasible set** as:

$$\tilde{\mathcal{D}} = \left\{ (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \mathcal{D} \left| \begin{array}{ll} c_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = 0, & \forall k \in E, \\ c_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0, & \forall k \in I \end{array} \right. \right\}. \quad (6.7)$$

Let $\mathbf{w} = \{w_1, \dots, w_m\}$ be a vector of m non-negative weights, comprising symbolic parameters and symbolic parameters specified by a neural network. Without loss of generality, assume the deep hinge-loss potentials are ordered such that the first m_{sy} potentials correspond to symbolic parameters, and the remaining m_{nn} potentials correspond to symbolic parameters specified by a neural network, where $m = m_{sy} + m_{nn}$. The weight vector can be expressed as:

$$\mathbf{w} = [\mathbf{w}_{sy}, \mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn})], \quad (6.8)$$

Define the **deep hinge-loss energy function** E as:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})). \quad (6.9)$$

A **deep hinge-loss Markov random field** P over variables \mathbf{y} , conditioned on symbolic inputs \mathbf{x}_{sy} and neural inputs \mathbf{x}_{nn} , is a probability density defined as follow:

$$P(\mathbf{y}|\mathbf{x}_{sy}, \mathbf{x}_{nn}) = \begin{cases} \frac{1}{Z(\mathbf{w}, \mathbf{x}_{sy}, \mathbf{x}_{nn})} \exp(-E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})), & (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ 0, & \text{otherwise,} \end{cases} \quad (6.10)$$

where the partition function $Z(\mathbf{w}, \mathbf{x}_{sy}, \mathbf{x}_{nn})$ is:

$$Z(\mathbf{w}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) = \int_{\mathbf{y} | (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}} \exp(-E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})) d\mathbf{y}. \quad (6.11)$$

Deep HL-MRFs naturally subsume traditional HL-MRFs. When neural networks \mathbf{g}_{nn} are not utilized to define parameters or random variables, the framework simplifies to the standard HL-MRF formulation. By incorporating neural networks, Deep HL-MRFs enhance the ability to model complex, non-linear dependencies beyond the scope of traditional linear potentials. Despite this added expressiveness, the inference objective remains convex with respect to the neural network predictions, ensuring efficient and scalable optimization. This makes Deep HL-MRFs a powerful tool for probabilistic reasoning in complex, high-dimensional domains. In the next section, I will introduce inference and learning over deep HL-MRFs, and in the subsequent chapter, I will introduce *Neural Probabilistic Soft Logic*, an extension of the Probabilistic Soft Logic framework, which provides a flexible way to define these distributions.

6.2 Inference and Learning in Deep Hinge-Loss Markov Random Fields

With the extension of traditional HL-MRFs to Deep HL-MRFs now established, this section formalizes the integration of Deep HL-MRFs into the NeSy-EBM framework, enabling the use of inference and learning methods introduced in the previous chapter.

This formulation depends on the underlying inference process, and for this work, I study the Maximum a posteriori (MAP) inference over an extended-value Deep HL-MRF energy function. MAP inference is the most common inference task in HL-MRF theory and remains central in Deep HL-MRFs, as it is required for parameter updates during the learning process. A direction for future research would be to formulate Deep HL-MRFs into the NeSy-EBM framework by studying from the perspective of marginal inference. This section proceeds as follows: it first provides a formal definition of the MAP inference problem, followed by a smooth reformulation to enable the full range of NeSy-EBM learning techniques and accommodate integer constraints on target variables. Finally, it presents a formal definition of the learning process for Deep HL-MRFs, framed within the NeSy-EBM learning framework.

6.2.1 MAP Inference

The primary inference task in both traditional HL-MRFs and Deep HL-MRFs is Maximum a Posteriori (MAP) inference. In Deep HL-MRFs, MAP inference seeks to find the most probable assignment to the free variables \mathbf{y} , given the observed symbolic inputs \mathbf{x}_{sy} and neural outputs $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$. This MAP formulation plays a central role in Deep HL-MRFs, supporting prediction tasks and acting as a key component in the iterative updates required for learning.

Deep HL-MRF energy functions can be interpreted as NeSy-EBMs by transforming them into extended-value energy functions that enforce the constraints defining the feasible set. Specifically, the energy function assigns infinite energy to configurations that fall outside the Deep HL-MRF feasible set and finite energy within this set, as follows:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \text{if } (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ \infty & \text{otherwise,} \end{cases}$$

where $\tilde{\mathcal{D}}$ denotes the feasible set, defined by the model’s constraints. The weight vector $\mathbf{w} = \{w_1, \dots, w_m\}$ is composed of both symbolic weights \mathbf{w}_{sy} and neural-defined symbolic

weights $\mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$:

$$\mathbf{w} = [\mathbf{w}_{sy}, \mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn})].$$

In the context of Deep HL-MRFs, MAP inference can be understood as the process of finding the MAP state of the conditional distribution defined by the deep HL-MRF model. Since the partition function $Z(\mathbf{w}, \mathbf{x})$ remains constant over the target variables, the MAP objective simplifies to minimizing the negative log probability of the Deep HL-MRF joint distribution. Consequently, MAP inference reduces to minimizing the Deep HL-MRF energy function over the feasible set. This can be formulated as follows:

$$\begin{aligned} \arg \max_{\mathbf{y} \in \mathbb{R}^{n_y}} P(\mathbf{y} | \mathbf{x}_{sy}, \mathbf{x}_{nn}) &\equiv \arg \min_{\mathbf{y} \in \mathbb{R}^{n_y}} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) \\ &\equiv \arg \min_{\mathbf{y} \in \mathbb{R}^{n_y}} \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ \text{s.t. } &(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}. \end{aligned}$$

In this formulation, the non-smooth, convex nature of Deep HL-MRF potentials results in a non-smooth convex optimization problem as it is a sum of convex functions. Further, the feasible set $\tilde{\mathcal{D}}$ is a convex polyhedron making MAP inference a linearly constrained, non-smooth convex program. To enable the full range of learning techniques discussed in Section 4.3 and to naturally accommodate integer constraints on target variables—a feature often used in discrete problems to enforce hard logic semantics—a reformulation of the problem is necessary. While adding integer constraints disrupts the convexity of MAP inference, modern solvers can often find global solutions or high-quality approximations even at large scales. Thus, both the inclusion of integer constraints and the application of advanced learning techniques require a reformulation of MAP inference as a linearly constrained quadratic program (LCQP).

Smooth Formulation For completeness, I introduce the smooth formulation, but this work is primarily contributed by my colleague in [43]. This formulation is instrumental in establishing continuity and curvature properties of the energy minimizer and the optimal value function but will not be discussed here. In this reformulation, m slack variables

with lower bounds are introduced, along with $2 \cdot n_{\mathbf{y}} + m$ linear constraints to represent both target variable bounds and deep hinge-loss potentials. These constraints, along with variable bounds, deep hinge-loss potentials, and any additional constraints ($q \geq 0$), are assembled into a matrix \mathbf{A} of dimension $(2 \cdot n_{\mathbf{y}} + q + 2 \cdot m) \times (n_{\mathbf{y}} + m)$ and an affine vector $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ with $2 \cdot n_{\mathbf{y}} + q + 2 \cdot m$ elements. The vector $\mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ captures the influence of neural predictions and symbolic inputs. Further, a positive semi-definite diagonal matrix $\mathbf{D}(\mathbf{w}_{sy})$ of size $(n_{\mathbf{y}} + m) \times (n_{\mathbf{y}} + m)$ and a vector $\mathbf{c}(\mathbf{w}_{sy})$ with $n_{\mathbf{y}} + m$ elements are defined using the symbolic weights to construct a quadratic objective. The original target variables and slack variables are then combined into a vector $\nu \in \mathbb{R}^{n_{\mathbf{y}} + m}$, resulting in the following regularized convex LCQP for MAP inference in Deep HL-MRFs:

$$\begin{aligned} V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) &:= \min_{\nu \in \mathbb{R}^{n_{\mathbf{y}} + m}} \nu^T (\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I}) \nu + \mathbf{c}(\mathbf{w}_{sy})^T \nu \quad (6.12) \\ \text{s.t. } \quad \mathbf{A}\nu + \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) &\leq 0, \end{aligned}$$

where $\epsilon \geq 0$ is a regularization parameter added to ensure strong convexity. The function $V(\mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ in (6.12) represents the optimal value of the LCQP-based MAP inference problem for Deep HL-MRFs.

By Slater's condition, strong duality holds when a feasible solution to (6.12) exists [18]. Thus, solving the dual problem provides an optimal solution to the primal problem. The dual problem of (6.12) is given by:

$$\begin{aligned} \min_{\mu \in \mathbb{R}_{\geq 0}^{2 \cdot (n_{\mathbf{y}} + m) + q}} \quad & h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \quad (6.13) \\ := \quad & \frac{1}{4} \mu^T \mathbf{A} (\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I})^{-1} \mathbf{A}^T \mu + \frac{1}{2} (\mathbf{A} (\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I})^{-1} \mathbf{c}(\mathbf{w}_{sy}) - 2 \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))^T \mu, \end{aligned}$$

where μ denotes the dual variables, and $h(\mu; \mathbf{w}_{sy}, \mathbf{b}(\mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})))$ is the dual objective. Given the diagonal nature of $\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I}$, inverting it is computationally efficient, making it practical to solve in the dual space and convert back to the primal solution.

The dual-to-primal variable mapping is:

$$\nu \leftarrow -\frac{1}{2} (\mathbf{D}(\mathbf{w}_{sy}) + \epsilon \mathbf{I})^{-1} (\mathbf{A}^T \mu + \mathbf{c}(\mathbf{w}_{sy})). \quad (6.14)$$

In contrast, mapping from primal to dual requires computing a pseudo-inverse of \mathbf{A} , which is more computationally intensive.

Solvers When using the original formulation without integer constraints, standard optimization solvers for HL-MRFs, such as the Alternating Direction Method of Multipliers (ADMM) or direct gradient descent methods [11], can be effectively applied to Deep HL-MRFs. Furthermore, for cases where Deep HL-MRFs are reformulated as LCQPs, off-the-shelf solvers, such as Gurobi, provide efficient solutions and also support additional practical constraints, including integer constraints.

6.2.2 Learning

With the MAP inference objective established in the previous subsection, the learning algorithms introduced in the previous chapter can be readily applied to Deep HL-MRFs within the NeSy-EBM framework. This subsection formalizes the learning objective for Deep HL-MRFs using an energy-based loss function that integrates symbolic and neural components.

Given a training dataset \mathcal{S} , which consists of P samples, each containing observed symbolic and neural data as well as target values, we define the dataset as follows:

$$\mathcal{S} := \{(\mathbf{t}_y^1, \mathbf{t}_{nn}^1, \mathbf{z}_{nn}^1, \mathbf{z}_y^1, \mathbf{x}_{sy}^1, \mathbf{x}_{nn}^1), \dots, (\mathbf{t}_y^P, \mathbf{t}_{nn}^P, \mathbf{z}_{nn}^P, \mathbf{z}_y^P, \mathbf{x}_{sy}^P, \mathbf{x}_{nn}^P)\},$$

The learning loss for Deep HL-MRFs can then be defined as a combination of neural, value-based, and minimizer-based losses:

$$\begin{aligned} L^i(E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) & \quad (6.15) \\ &= L_{NN}(\mathbf{g}_{nn}^i(\mathbf{x}_{nn}^i, \mathbf{w}_{nn}), \mathbf{t}_{nn}^i) \quad \text{Neural} \\ &\quad + L_{Val}(E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \quad \text{Value-Based} \\ &\quad + L_{Min}(E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) \quad \text{Minimizer-Based} \end{aligned}$$

where $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ is defined as the extended-value energy function:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \text{if } (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ \infty & \text{otherwise,} \end{cases}$$

with $\tilde{\mathcal{D}}$ representing the feasible set defined by the model’s constraints, ensuring finite energy values only within the feasible set. The complete learning problem for Deep HL-MRFs is then just:

$$\begin{aligned} & \arg \min_{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}} \mathcal{L}(E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \\ = & \arg \min_{(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \in \mathcal{W}_{sy} \times \mathcal{W}_{nn}} \frac{1}{P} \sum_{i=1}^P L^i(E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}), \mathcal{S}_i) + \mathcal{R}(\mathbf{w}_{sy}, \mathbf{w}_{nn}) \end{aligned}$$

Optimization The optimization of this learning objective can be approached through several methods, broadly discussed in the previous chapter’s learning section. These approaches fall into four primary categories: modular optimization, gradient descent, bilevel value-function optimization, and stochastic policy optimization. A detailed examination of the algorithms that implement these strategies was a significant contribution by my colleague Dickens (2024), expanding on foundational HL-MRF learning theory as presented by [11].

6.3 Syntax and Semantics of Neural Probabilistic Soft Logic

Neural Probabilistic Soft Logic is a declarative language designed to construct deep hinge-loss markov random fields within the neural-symbolic energy-based model framework. Like its predecessor, Probabilistic Soft Logic (PSL), NeuPSL defines core functionalities that should be supported by all implementations while maintaining the flexibility to accommodate future extensions. The syntax presented here is capable of representing a broad class of Deep HL-MRFs and NeSy modeling paradigms, however, as new challenges and domains arise, additional syntax may be introduced to simplify or extend the construction of various Deep HL-MRF models.

This section builds upon the foundational work of Bach et al. (2017) in defining PSL, extending it to include deep neural components for both parameters and random variables. Since NeuPSL generalizes PSL by encompassing neural components, a separate discussion of PSL is unnecessary; PSL can be seen as a special case of NeuPSL where the neural components are omitted. It is important to note, as many components will be the

same as PSL, I refer readers interested in the foundational PSL program to [11] for further details.

Preliminaries Before diving into the formal syntax and semantics of NeuPSL, I provide a high-level definition:

Definition 14 (Neural Probabilistic Soft Logic Program). *A NeuPSL program is a set of rules that serve as templates for defining **deep hinge-loss potentials** or **deep hard constraints** within a Deep HL-MRF. These rules, when instantiated over a base of ground atoms (from a database) and deep ground atoms (from a neural networks’ output), induce a Deep HL-MRF conditioned on specified observations. The parameters of these instantiated rules are defined with either a symbolic parameter or through neural network-derived symbolic parameter.*

Informally, a NeuPSL program is grounded or instantiated over data and neural model outputs, so the universe over which to ground must be defined. A NeuPSL program is grounded using elements in a domain of discourse known as a *constant*. Constants can be entities or attributes, such as the constant “*cat1*” can denote a cat, “*Ashton*” can denote a cat’s name, and “*13*” can denote a cat’s age. When defining a NeuPSL program, instead of specifying constants everywhere, a placeholder or *variable* can be defined that will be replaced with constants during grounding. For instance a placeholder variable for all cats in a database could be “*cats*”. A *term* (either a constant or variable) are connected together with a *predicate*. For example, a *SeniorCat* is a binary predicate, i.e., taking in two arguments, which represents whether a cat given its age is a senior. Finally, predicates and terms combined make an *atom* and NeuPSL extends this also to include *deep atoms* or atoms defined by a deep model. Formally,

Definition 15 (Atom, Deep Atom, Ground Atom). *Let \mathcal{C} be the set of constants, representing fixed entities or values within the domain of discourse, and \mathcal{V} be the set of variables, which serve as placeholders that can be substituted by constants during grounding. A **term** t is an element of either \mathcal{C} or \mathcal{V} .*

A **predicate** P is defined by a unique identifier and an arity $k \in \mathbb{N}$, representing a relation over k terms:

$$P : (\mathcal{C} \cup \mathcal{V})^k \rightarrow [0, 1],$$

where P maps a sequence of k terms (t_1, \dots, t_k) , with $t_i \in \mathcal{C} \cup \mathcal{V}$, to a continuous truth value in $[0, 1]$.

An **atom** A is a predicate applied to a specific sequence of terms:

$$A = P(t_1, \dots, t_k).$$

A **deep atom** A_d extends the standard atom by incorporating parameters from a neural network:

$$A_d = P(t_1, \dots, t_k; \mathbf{w}_{nn}),$$

where \mathbf{w}_{nn} are the neural network weights influencing the predicate.

A **ground atom** A_g is an atom where all terms are constants:

$$A_g = P(c_1, \dots, c_k), \quad c_i \in \mathcal{C} \text{ for all } i.$$

A NeuPSL program is then defined as a set of templated rules that are relations or operations over atoms.

Definition 16 (Rules). A **rule** in a NeuPSL program is a function r that maps a set of atoms, ground atoms, or deep atoms to the interval $[0, 1]$. Formally:

$$r : \{A_1, A_2, \dots, A_n\} \rightarrow [0, 1],$$

where each A_i is an atom, ground atom, or deep atom.

Rules in NeuPSL can be either **soft** or **hard**. A soft rule has a finite weight $w \in [0, \infty)$, while a hard rule, on the other hand, is a strict constraint with an infinite weight, meaning it must always be satisfied. Furthermore, rules can either be logical or arithmetic:

1. **Logical Rules:** These express probabilistic dependencies in the form:

$$r : w \left(\bigwedge_{i=1}^k L_i \rightarrow \bigvee_{j=1}^m L_j \right),$$

where:

- w is the weight of the rule.
- L_i and L_j are literals (either atoms or negated atoms).
- \bigwedge denotes a conjunction (AND), and \bigvee denotes a disjunction (OR).

2. **Arithmetic Rules:** These relate linear combinations of atoms via an inequality or equality. Formally, an arithmetic rule is of the form:

$$r : w \left(\sum_{i=1}^a \alpha_i A_i \triangle \sum_{j=1}^b \beta_j B_j \right),$$

where:

- w is the weight of the rule.
- α_i, β_j are coefficients.
- A_i, B_j are atoms.
- \triangle represents either $=$ (equality) or \leq (inequality).

Note 6.3.1. NeuPSL employs Lukasiewicz logic for its logical rules to maintain a convex function that is also differentiable. The Lukasiewicz t -norm and t -co-norm are \wedge and \vee operators that correspond to the Boolean logic operators for integer inputs (along with the negation operator \neg):

$$A \wedge B = \max(0, A + B - 1),$$

$$A \vee B = \min(1, A + B),$$

$$\neg A = 1 - A,$$

where A and B are truth values of atoms or expressions over atoms.

Grounding, or instantiation, is the process by which abstract rules, defined in terms of predicates and variables, are translated into concrete constraints or potentials over specific entities in the domain. While a comprehensive explanation of grounding is beyond the scope of this section, readers seeking a detailed formal description are referred to [11]. It is, however, important to distinguish how grounding is affected by the integration of neural networks. In brief, the grounding process with the inclusion of neural predictions operates as follows:

1. **Instantiation:** Variables in the NeuPSL rules are substituted with constants from the domain, generating all possible ground atoms and instances for each rule. During this step, a ground rule is converted into deep hinge-loss potential. For example, logical rules are translated using Łukasiewicz logic.
2. **Ground Atom Association:** After instantiation, each ground atom is linked to either a value from a database or an output from a neural network. For neural outputs, a specialized mapping is created to associate terms in the ground atoms with the corresponding neural features.
3. **Variable Assignment:**
 - **Observed Variables (\mathbf{x}_{sy}):** Ground atoms with values specified in the database are assigned as observed variables.
 - **Free Variables (\mathbf{y}):** Ground atoms not specified in the database are initialized as free variables with default or random values.
 - **Neural Assignment:** For deep atoms and parameters, input features are processed through the corresponding neural networks \mathbf{g}_{nn} . The neural network outputs are then assigned to these atoms or parameters.

A detailed description of the system design, including how the assignment of terms to neural features is handled, will be provided in the following section. Consider the following example of a NeuPSL program, along with the grounding process:

Example 6.3.1 (Species Classification in NeuPSL). *Consider a NeuPSL program designed for species classification, which leverages additional knowledge about whether two images depict the same underlying entity. This integration allows for refining predictions, especially when an image is of lower quality, by utilizing information from associated images of the same entity.*

NeuPSL Program: *A NeuPSL program is a declarative framework that generates a Deep HL-MRF by grounding symbolic templated rules and neural network architectures with a symbolic database and mappings from atoms to neural features.*

Symbolic Rules:

$$\begin{aligned}
 w : \text{NEURAL}(\text{Img}_1, \text{Species}) \wedge \text{SAMEENTITY}(\text{Img}_1, \text{Img}_2) \\
 \rightarrow \text{CLASS}(\text{Img}_2, \text{Species}) \\
 \text{CLASS}(\text{Img}, \text{Species}+) = 1.
 \end{aligned}$$

Neural Architectures:

$$g_{CNN} : \mathcal{X}_{nn} \rightarrow [0, 1]^3$$

Input: Image

Output: Classification scores for $\{\text{Cat}, \text{Dog}, \text{Frog}\}$

Atom to Neural Feature Mapping:

$$\text{NEURAL}(\text{ImgA}, \text{Cat}) =$$





$$\text{NEURAL}(\text{ImgA}, \text{Dog}) =$$




$$\text{NEURAL}(\text{ImgA}, \text{Frog}) =$$



$$\text{NEURAL}(\text{ImgB}, \text{Cat}) =$$


$$\text{NEURAL}(\text{ImgB}, \text{Dog}) =$$


$$\text{NEURAL}(\text{ImgB}, \text{Frog}) =$$


Symbolic Database:

$$\text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) = 1$$

$$\text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) = 1$$

$$\text{NEURAL}(\text{ImgA}, \text{Cat})$$

$$\text{NEURAL}(\text{ImgA}, \text{Dog})$$

$$\text{NEURAL}(\text{ImgA}, \text{Frog})$$

$$\text{NEURAL}(\text{ImgB}, \text{Cat})$$

$$\text{NEURAL}(\text{ImgB}, \text{Dog})$$

$$\text{NEURAL}(\text{ImgB}, \text{Frog})$$

$$\text{CLASS}(\text{ImgA}, \text{Cat})$$

$$\text{CLASS}(\text{ImgA}, \text{Dog})$$

$$\text{CLASS}(\text{ImgA}, \text{Frog})$$

$$\text{CLASS}(\text{ImgB}, \text{Cat})$$

$$\text{CLASS}(\text{ImgB}, \text{Dog})$$

$$\text{CLASS}(\text{ImgB}, \text{Frog})$$

Note 6.3.2. *In practice, a closed-world assumption is applied, meaning that any predicate not explicitly defined in the symbolic database is assumed to have a value of zero. This assumption significantly accelerates the rule instantiation process, as it allows many potential*

grounded rules to be trivially evaluated and ignored. Furthermore, the atom-to-neural feature mapping can be simplified by assuming that the final term in the atom corresponds to the neural network’s output. This reduces the mapping complexity, enabling a more concise representation such as: $\text{NEURAL}(\text{ImgA}, \text{Species}) = \text{imgA}$ and $\text{NEURAL}(\text{ImgB}, \text{Species}) = \text{imgB}$.

Given the above data, templated rules, and neural architecture, the instantiation or grounding process begins by substituting every valid combination of constants into the rules. This generates the following ground rules:

$$\begin{aligned}
&\text{NEURAL}(\text{ImgA}, \text{Cat}) \wedge \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) \rightarrow \text{CLASS}(\text{ImgB}, \text{Cat}) \\
&\text{NEURAL}(\text{ImgA}, \text{Dog}) \wedge \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) \rightarrow \text{CLASS}(\text{ImgB}, \text{Dog}) \\
&\text{NEURAL}(\text{ImgA}, \text{Frog}) \wedge \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) \rightarrow \text{CLASS}(\text{ImgB}, \text{Frog}) \\
&\text{NEURAL}(\text{ImgB}, \text{Cat}) \wedge \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) \rightarrow \text{CLASS}(\text{ImgA}, \text{Cat}) \\
&\text{NEURAL}(\text{ImgB}, \text{Dog}) \wedge \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) \rightarrow \text{CLASS}(\text{ImgA}, \text{Dog}) \\
&\text{NEURAL}(\text{ImgB}, \text{Frog}) \wedge \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) \rightarrow \text{CLASS}(\text{ImgA}, \text{Frog}) \\
&\text{CLASS}(\text{ImgA}, \text{Cat}) + \text{CLASS}(\text{ImgA}, \text{Dog}) + \text{CLASS}(\text{ImgA}, \text{Frog}) = 1 \\
&\text{CLASS}(\text{ImgB}, \text{Cat}) + \text{CLASS}(\text{ImgB}, \text{Dog}) + \text{CLASS}(\text{ImgB}, \text{Frog}) = 1
\end{aligned}$$

Notice that invalid substitutions, such as:

$$\begin{aligned}
&\text{NEURAL}(\text{ImgA}, \text{Cat}) \wedge \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) \rightarrow \text{CLASS}(\text{ImgB}, \text{Dog}), \\
&\text{NEURAL}(\text{ImgA}, \text{Cat}) \wedge \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) \rightarrow \text{CLASS}(\text{ImgB}, \text{Frog}),
\end{aligned}$$

are not created. This is because variable substitution is consistently applied across all instances of the same variable within the rule, ensuring logical consistency. For example, the variable *Species* is consistently substituted with the same constant across the rule. Grounding in NeuPSL involves significant computational optimization, which allows efficient instantiation of rules over large datasets. For a detailed discussion on optimizing this process, refer to [11]. The next step in instantiation is to transform each ground rule into

its corresponding hinge-loss potential and converted into a weighted sum.

$$\begin{aligned}
& w * \max\{0, \text{NEURAL}(\text{ImgA}, \text{Cat}) + \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) - 1 - \text{CLASS}(\text{ImgB}, \text{Cat})\} + \\
& w * \max\{0, \text{NEURAL}(\text{ImgA}, \text{Dog}) + \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) - 1 - \text{CLASS}(\text{ImgB}, \text{Dog})\} + \\
& w * \max\{0, \text{NEURAL}(\text{ImgA}, \text{Frog}) + \text{SAMEENTITY}(\text{ImgA}, \text{ImgB}) - 1 - \text{CLASS}(\text{ImgB}, \text{Frog})\} + \\
& w * \max\{0, \text{NEURAL}(\text{ImgB}, \text{Cat}) + \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) - 1 - \text{CLASS}(\text{ImgA}, \text{Cat})\} + \\
& w * \max\{0, \text{NEURAL}(\text{ImgB}, \text{Dog}) + \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) - 1 - \text{CLASS}(\text{ImgA}, \text{Dog})\} + \\
& w * \max\{0, \text{NEURAL}(\text{ImgB}, \text{Frog}) + \text{SAMEENTITY}(\text{ImgB}, \text{ImgA}) - 1 - \text{CLASS}(\text{ImgA}, \text{Frog})\}
\end{aligned}$$

The feasible set is determined by the instantiated unweighted rules, ensuring that the classification probabilities for each image sum to one:

$$\begin{aligned}
& \max\{0, \text{CLASS}(\text{ImgA}, \text{Cat}) + \text{CLASS}(\text{ImgA}, \text{Dog}) + \text{CLASS}(\text{ImgA}, \text{Frog}) - 1\}, \\
& \max\{0, -\text{CLASS}(\text{ImgA}, \text{Cat}) - \text{CLASS}(\text{ImgA}, \text{Dog}) - \text{CLASS}(\text{ImgA}, \text{Frog}) + 1\}, \\
& \max\{0, \text{CLASS}(\text{ImgB}, \text{Cat}) + \text{CLASS}(\text{ImgB}, \text{Dog}) + \text{CLASS}(\text{ImgB}, \text{Frog}) - 1\}, \\
& \max\{0, -\text{CLASS}(\text{ImgB}, \text{Cat}) - \text{CLASS}(\text{ImgB}, \text{Dog}) - \text{CLASS}(\text{ImgB}, \text{Frog}) + 1\}.
\end{aligned}$$

Each ground atom is linked to either a value from the symbolic database or the output of a neural model. This requires a forward pass through the neural network to obtain initial predictions. Suppose the neural model outputs the following values for each image:

$$\begin{aligned}
& \text{NEURAL}(\text{ImgA}, \text{Cat}) = 0.7, \\
& \text{NEURAL}(\text{ImgA}, \text{Dog}) = 0.2, \\
& \text{NEURAL}(\text{ImgA}, \text{Frog}) = 0.1, \\
& \text{NEURAL}(\text{ImgB}, \text{Cat}) = 0.6, \\
& \text{NEURAL}(\text{ImgB}, \text{Dog}) = 0.3, \\
& \text{NEURAL}(\text{ImgB}, \text{Frog}) = 0.1.
\end{aligned}$$

All unobserved variables, referred to as open variables, are initially assigned a value (e.g.,

0.5):

$$\begin{aligned}
&\text{CLASS}(\text{ImgA}, \text{Cat}) = 0.5, \\
&\text{CLASS}(\text{ImgA}, \text{Dog}) = 0.5, \\
&\text{CLASS}(\text{ImgA}, \text{Frog}) = 0.5, \\
&\text{CLASS}(\text{ImgB}, \text{Cat}) = 0.5, \\
&\text{CLASS}(\text{ImgB}, \text{Dog}) = 0.5, \\
&\text{CLASS}(\text{ImgB}, \text{Frog}) = 0.5.
\end{aligned}$$

With the deep HL-MRF now fully grounded, where each ground atom has an associated random variable and value assignment, the system is prepared for inference or learning.

6.4 Defining NeSy-EBM Modeling Paradigms using NeuPSL

With the foundational syntax and semantics of NeuPSL established in the previous section, this section will explore how NeuPSL enables the construction of the three core modeling paradigms, as introduced in Section 3.2. These paradigms, which include deep variables, deep parameters, and deep potentials, can be systematically expressed within NeuPSL, demonstrating the framework’s flexibility in defining complex NeSy-EBM models. Importantly, while NeuPSL can support a broader range of NeSy-EBM paradigms, this section highlights the simplicity and adaptability with which it can define them.

6.4.1 Deep Variables

The first NeuPSL modeling paradigm, termed *deep variables*, represents a deep symbolic variable (Section 3.2.1) NeSy-EBM modeling approach. In this paradigm, NeuPSL incorporates *deep atoms*—atoms parameterized by neural network outputs—while retaining symbolic weights for each rule. The key characteristic of this setup is that the neural-predicted variables are “fixed” or directly linked to a latent or target variable within the Deep HL-MRF energy function.

Formally, in the deep variables paradigm, the symbolic potential set is a singleton, denoted by $\Psi = \{\psi\}$, with an index set $\mathbf{J}_\Psi = \{1\}$ such that $\Psi_1 = \psi$. This paradigm enforces consistency between the free variables \mathbf{y} and the neural network outputs $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ through an indicator function $I_{\mathbf{y}}$ defined as:

$$I_{\mathbf{y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \begin{cases} 0 & \mathbf{y}_i = [\mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]_i, \quad \forall i \in \{1, \dots, d_{nn}\}, \\ \infty & \text{otherwise,} \end{cases} \quad (6.16)$$

where \mathbf{y}_i and $[\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]_i$ represent the i -th component of the variable vector \mathbf{y} and the neural output vector $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, respectively.

The symbolic component of the NeSy-EBM model is then formulated as:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \sum_{j=1}^m w_{sy,j} \phi_j(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn})) + I_{\mathbf{y}}(\mathbf{y}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), \quad (6.17)$$

where each $w_{sy,j}$ is a purely symbolic weight, independent of the neural predictions.

The extended-value energy function $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ for the deep variables paradigm is defined as:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \text{if } (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ \infty & \text{otherwise,} \end{cases}$$

where \mathbf{w}_{sy} represents the symbolic weights, i.e., $\mathbf{w} = [\mathbf{w}_{sy}]$.

Note 6.4.1. An alternative formulation would incorporate the indicator function $I_{\mathbf{y}}$ as a hard constraint, effectively expanding the feasible set $\tilde{\mathcal{D}}$ to:

$$\hat{\mathcal{D}} = \left\{ (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \mathcal{D} \left| \begin{array}{l} c_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) = 0, \quad \forall k \in E, \\ c_k(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \leq 0, \quad \forall k \in I, \\ \mathbf{y}_i - \mathbf{g}_{nn}^{RV}(\mathbf{x}_{nn}, \mathbf{w}_{nn})_i = 0, \quad \forall i \in \{1, \dots, d_{nn}\}. \end{array} \right. \right\}. \quad (6.18)$$

In practice, this modeling paradigm is implemented by treating the neural model's predictions as observed variables during MAP optimization.

6.4.2 Deep Weights

The second NeuPSL modeling paradigm, termed *deep weights*, defines a deep symbolic parameter (Section 3.2.2) NeSy-EBM modeling approach. In this approach, NeuPSL incorporates neural network outputs directly as weights within symbolic rules without the inclusion of deep atoms. A key distinction in this paradigm is that neural-derived weights are assigned on a per-grounded-rule basis, unlike traditional NeuPSL, where weights are shared across all groundings of a templated rule. This per-grounded-rule weighting enhances expressivity by enabling context-sensitive weighting of rules, albeit at the cost of increased model complexity and parameter count.

Formally, in this paradigm, the symbolic potential set is a singleton, denoted $\Psi = \{\psi\}$, with a trivial index set $\mathbf{J}_\Psi = \{1\}$ such that $\Psi_1 = \psi$. The weights for each potential are defined either by neural network outputs, $\mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$, or by symbolic weights, \mathbf{w}_{sy} , yielding:

$$\mathbf{w} := [\mathbf{w}_{sy}, \mathbf{g}_{nn}^{Param}(\mathbf{x}_{nn}, \mathbf{w}_{nn})]. \quad (6.19)$$

The symbolic component of the NeSy-EBM model under the deep weights paradigm is expressed as:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \sum_{j=1}^m w_j \phi_j(\mathbf{y}, \mathbf{x}_{sy}), \quad (6.20)$$

where w_j represents the neural or symbolic weights assigned to each potential ϕ_j .

The corresponding extended-value energy function $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ is then defined as:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \text{if } (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ \infty & \text{otherwise.} \end{cases}$$

6.4.3 Deep Rules

The final NeuPSL modeling paradigm, termed *deep rules*, aligns with the deep symbolic potentials (Section 3.2.3) NeSy-EBM modeling approach. In this paradigm, the

neural model dynamically selects a subset of potential functions (rules) that will be used to answer a specific query. Unlike the previous paradigms, deep rules do not involve deep atoms or neural-derived weights; instead, they focus on adapting the symbolic rule set based on neural predictions, allowing the model to adjust its reasoning structure in response to the neural model’s interpretation.

Formally, in the deep rules paradigm, the symbolic potential set Ψ represents the complete set of all potential functions that NeuPSL can instantiate. The index set for these potentials is driven by the neural component’s output, denoted as $\mathbf{J}_\Psi = \text{Range}(\mathbf{g}_{nn})$. Each specific neural output $\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$ selects a subset of potential functions $\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}$, defining a rule set conditioned on the neural model’s interpretation of the input.

The variable and parameter domains of each selected symbolic potential are given by $V_\psi = \mathcal{Y} \times \mathcal{X}_{sy}$ and $\text{Params}_\psi = \mathcal{W}_{sy}$, respectively. Thus, the symbolic component g_{sy} of the NeSy-EBM model in the deep rules paradigm can be expressed as:

$$g_{sy}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) := \sum_{j=1}^{|\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}|} w_j \Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}),j}(\mathbf{y}, \mathbf{x}_{sy}),$$

where each $\Psi_{\mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}),j}$ represents a potential function chosen based on the neural output, and w_j denotes the symbolic weight associated with each selected potential.

The extended-value energy function $E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn})$ incorporates this selection mechanism as follows:

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{sy}, \mathbf{w}_{nn}) = \begin{cases} \sum_{g_{sy}(\cdot) \in \Psi} g_{sy}(\cdot) \cdot \delta_{j, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})} & \text{if } (\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}) \in \tilde{\mathcal{D}}, \\ \infty & \text{otherwise,} \end{cases}$$

where $\delta_{j, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})}$ is an indicator function that takes the value 1 if the neural model selects the potential ϕ_j and 0 otherwise. This setup ensures that only the selected potentials contribute to the energy function, while unselected potentials have no effect.

Note 6.4.2. *In practice, calculating the full potential set Ψ is computationally infeasible due to its scale. To approximate the deep rules paradigm, a generative approach is often employed, where a large language model (LLM) is queried to dynamically generate a symbolic program based on the neural model’s interpretation. This approach aligns the potential*

set with the neural model’s predictions in real time, creating a flexible and context-sensitive symbolic rule set.

6.5 NeuPSL System

As described in the previous sections, NeuPSL is a probabilistic programming framework designed to define and reason over Deep HL-MRFs. However, its practical implementation involves significant complexity due to the integration of symbolic and neural systems. This complexity arises particularly from combining PSL, implemented in Java [59], with neural frameworks typically based in Python [109], such as PyTorch [100], TensorFlow [1], and JAX [19]. This section delves into the system-level details of NeuPSL, focusing on its architecture, shared memory communication, and the operations necessary for seamless integration between symbolic and neural components. First, I provide a high-level overview of the system workflow (Section 6.5.1). Then, I describe the shared memory mechanism and its associated functions (Section 6.5.2). For additional details on the implementation, I encourage the reader to look at the codebase: <https://github.com/linqs/psl>.

6.5.1 System-Level Workflow

NeuPSL’s workflow integrates neural and symbolic reasoning components through *initialization*, *inference and learning*, and *finalization*:

1. **Initialization:** The initialization phase begins with the symbolic component. NeuPSL initializes a shared memory structure that facilitates communication between the neural and symbolic components during both inference and learning. Next, the NeuPSL system is configured to locate the neural model implementation through a provided Python wrapper class. NeuPSL creates a Python instance, which initializes the neural models required for the specific task and connects them to the shared memory. The Python neural model then maps symbolic atoms from the NeuPSL program to their corresponding data sources. It performs an initial prediction pass, generating preliminary values for the symbolic model. These values are written to

the shared memory, and the Python model signals the Java-based symbolic component that it is ready to proceed. The symbolic model subsequently completes its grounding process. It reads the initial predictions from the neural model and assigns these values to the appropriate ground atoms within the symbolic program. At this stage, the neural-symbolic system is fully initialized and ready to perform inference or learning.

2. **Inference and Learning:** Inference and learning in NeuPSL are designed as iterative processes, alternating between the neural model generating predictions and fitting and the symbolic model performing optimization. For inference, once the neural model has made its predictions (either during initialization or the previous batched gradient step), the symbolic model performs standard Maximum A Posteriori (MAP) optimization over its target and latent variables. After the optimization is complete, the results are either written in an output file or stored in shared memory for subsequent steps (if the system is in the learning phase).

In learning, inference acts as a subprocess, iterating through repeated steps of symbolic optimization. After the symbolic model completes its optimization, gradients for the neural variables are computed and written to the shared memory. The Python (neural) instance is then signaled to perform a fitting step, which updates the neural model parameters based on the received symbolic gradients. During this step, the symbolic gradient is combined with the neural model’s loss, weighted by a parameter α that balances the influence of each gradient source. For example, $\alpha = 0.5$ equally weights the neural and symbolic gradients, whereas other values can prioritize one over the other. After completing the fitting step, the neural model generates updated predictions by performing another forward pass. It writes the new predictions to the shared memory, enabling the symbolic model to begin a new round of optimization. This iterative loop continues until a convergence criterion or stopping condition is met.

The system also incorporates support for stochastic gradient descent (SGD) and batching to handle larger datasets effectively. While details of batch processing are

not fully described here, the design accounts for challenges such as synchronizing epochs between the neural and symbolic components, managing shared gradients, and ensuring efficient memory usage across iterations. These features ensure that NeuPSL can scale to practical, large-scale applications while maintaining tight integration between neural and symbolic reasoning.

3. **Finalization:** Upon completing the training or inference process, the system saves the final model parameters, clears or reinitializes the shared memory, and prepares for subsequent tasks, if needed.

6.5.2 Shared Memory Mechanism

Integrating the Java-based PSL codebase with Python-based neural frameworks presents inherent complexities. To address this, NeuPSL employs a shared memory mechanism that facilitates seamless communication between symbolic and neural systems. This design eliminates the need to rewrite the PSL system entirely in Python, preserving its established codebase while enabling efficient data exchange between the two frameworks.

The shared memory buffer is segmented, with the first segment allocated for metadata. This metadata contains essential details, such as the number of bytes to be read or written. Both the symbolic and neural components are aware of the memory layout, including specific byte offsets corresponding to particular ground atoms.

6.5.2.1 Python Side: Neural Interface

- **init:** Initializes the shared memory and loads the neural architecture.
- **write_data:** Writes neural outputs, such as predictions or observed variable values, to shared memory.
- **read_data:** Reads symbolic gradients or label information from shared memory to inform neural parameter updates.

6.5.2.2 Java Side: Symbolic Interface

- **init:** Initializes shared memory and establishes communication with the Python-based neural module.
- **write_data:** Writes symbolic outputs, such as gradients or label information, to shared memory for use by the neural module.
- **read_data:** Reads neural predictions or random variable values from shared memory to incorporate into symbolic reasoning tasks.

Chapter 7

Empirical Analysis

Throughout this dissertation, I have laid the foundational framework for principled neural-symbolic AI by introducing a cohesive set of architectural axioms (Chapter 2), a universal mathematical language (Section 3), and a comprehensive collection of design principles (Chapter 4 and Chapter 5). Building upon this foundation, in the previous chapter (Chapter 6), I proposed Neural Probabilistic Soft Logic a novel and general NeSy system capable of handling a wide range of these foundational components. This chapter now seeks to empirically evaluate NeuPSL’s capabilities, demonstrate its alignment with the proposed foundations, showcase its distinctions from other prominent NeSy systems, and explore common pitfalls in NeSy inference and learning, along with mitigation strategies.

To achieve these objectives, the first portion of this experimental evaluation focuses on NeuPSL’s performance across various neural-symbolic inference and learning settings. This aims to demonstrate not only the versatility and generality NeuPSL provides as a framework of NeuPSL but also to highlight the use cases for NeSy approaches. The second portion of this section evaluates how NeuPSL compares to other prominent NeSy systems on canonical tasks, providing insights into where and when these systems are most effective. Finally, the last portion of this chapter intentionally highlights pitfalls encountered in NeSy inference and learning, along with corresponding mitigation strategies, offering a roadmap for addressing these challenges and advancing the field. Through this empirical

analysis, I aim to validate NeuPSL as a robust and scalable NeSy approach while drawing broader insights into the strengths, limitations, and future directions of neural-symbolic integration. In short, this experimental evaluation aims to answer the following research questions:

- **RQ1:** Can NeSy methods improve the accuracy and reasoning capabilities of deep learning models, particularly within the inference and learning pipelines?
- **RQ2:** Under what conditions do principled NeSy approaches, especially NeuPSL, demonstrate optimal performance?
- **RQ3:** How effective are the proposed mitigation strategies in addressing specific pitfalls that arise during NeSy inference and learning?
- **RQ4:** How does the introduction of symbolic prior knowledge through NeuPSL affect the internal representations of a deep learning model?
- **RQ5:** How does the fine-tuning of subsymbolic methods with symbolic information influence their generalization to unseen data?

The empirical analysis is organized into four main sections: First, Section 7.1 introduces the datasets used in the experiments and provides an overview of the neural-symbolic models employed throughout the evaluation. Next, Section 7.2 examines the application of NeuPSL across a variety of critical NeSy tasks. This section is divided into inference tasks, including constraint satisfaction, joint reasoning, and few-shot reasoning, as well as learning tasks, such as decomposed task learning, semi-supervised learning, and zero-shot learning. Following this, Section 7.3 presents a comparative analysis of NeuPSL alongside four prominent NeSy frameworks. Finally, Section 7.4 investigates a selection of the most prominent NeSy pitfalls, with a particular focus on challenges specific to the NeuPSL formulation.

Note 7.0.1. *Some of the experiments presented here were primarily conducted by my colleagues. My role included editing the paper, discussing potential improvements, and offering insights into problem-solving and drawing conclusions. These experiments include*

all pathfinding results [43], zero-shot object navigation results [145], and semi-supervised learning experiments [43]. While my direct contributions to these specific results were more limited, including them here provides a more comprehensive and complete perspective on the overall findings. I encourage readers interested in a deeper exploration of these experiments are encouraged to consult the original papers.

7.1 Datasets and Models

This subsection introduces the NeSy datasets and models, which will be utilized throughout the empirical analysis. Moreover, any modifications made to answer specific research questions will be described in the following subsections. Additional details about the NeuPSL models can be found in the appendix (Appendix A).

- **MNIST-Add-k Dataset:** MNIST-Add- k is a canonical NeSy dataset introduced by Manhaeve et al. (2021) where models must determine the sum of each pair of digits from two lists of MNIST images. An MNIST-Add k equation consists of two lists of $k > 0$ MNIST images. For instance, $[\mathbf{3}] + [\mathbf{5}] = \mathbf{8}$ is an MNIST-Add1 equation, and $[\mathbf{0}, \mathbf{4}] + [\mathbf{3}, \mathbf{7}] = \mathbf{41}$ is an MNIST-Add2 equation.

Evaluation: For all experiments, we evaluate models over 5 splits of the low-data setting proposed by Manhaeve et al. (2021) with 600 total images for training and 1,000 images each for validation and test. Prediction performance in this setting is measured by the accuracy of the image classifications and the inferred sums. Constraint satisfaction consistency in this setting is the proportion of predictions that satisfy the semantics of addition.

Baseline Model: The baseline neural model for all MNIST-Add k datasets is a ResNet18 convolutional neural network backbone [62] with a 2-layer multi-layer perceptron (MLP) prediction head. The baseline is trained and applied as a digit classifier. Further, to allow the baseline to leverage the unlabeled training data in the semi-supervised settings, the digit classifier backbone is pre-trained using the SimCLR self-supervised learning framework [24]. Augmentations are used to obtain

positive pairs for the contrastive pre-training process.

NeuPSL Model: The NeuPSL model is a composition of the baseline digit classifier and a symbolic component created with NeuPSL that encodes the semantics of addition. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a decomposed task structure (Section 4.2.3), with the neural component predicting digits and the symbolic component predicting additions.

- **Visual-Sudoku Dataset:** Visual-Sudoku, first introduced by Wang et al. (2019), is a dataset containing a collection of 9×9 Sudoku puzzles constructed from MNIST images. In each puzzle, 30 cells are filled with MNIST images and are referred to as *clues*. The remaining cells are empty. The task is to correctly classify all clues and fill in the empty cells with digits that satisfy the rules of Sudoku: no repeated digits in any row, column, or box.

Evaluation: For all experiments, results are reported across 5 splits with 20 puzzles for training and 100 puzzles each for validation and test. There is an equal number of MNIST images (600) in the training datasets for Visual-Sudoku and MNIST-Addk. Prediction performance in this setting is measured by the accuracy of the image classifications. Constraint satisfaction consistency in this setting is the proportion of predictions that satisfy the rules of Sudoku.

Baseline Model: The baseline neural model for Visual-Sudoku is the same as that of the MNIST-Addk.

NeuPSL Model: The NeuPSL model is a composition of the baseline digit classifier and a symbolic component created with NeuPSL that encodes the rules of Sudoku. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a decomposed task structure (Section 4.2.3), with the neural component predicting digits and the symbolic component predicting the label for both squares with and without MNIST images.

- **Pathfinding Dataset:** Pathfinding is a NeSy dataset introduced by Vlastelica et al. (2020) consisting of 12000 randomly generated images of terrain maps from the

Warcraft II tileset. The images are partitioned into 12×12 grids where each vertex represents a terrain with a cost. The task is to find the lowest cost path from the top left to the bottom right corner of each image.

Evaluation: For all experiments, results are reported over 5 splits generated by partitioning the images into sets of 10,000 for training, 1,000 for validation, and 1,000 for testing. Prediction performance in this setting is measured by the proportion of valid predicted paths, i.e., continuous, and that have a minimum cost. Constraint satisfaction continuity in this setting is measured by the proportion of predictions with a continuous predicted path.

Baseline Model: The baseline neural model for the Pathfinding dataset is a ResNet18 convolutional neural network. The input of the ResNet18 path-finder baseline is the full Warcraft II map, and the output is the predicted shortest path. The model is trained using the labeled paths from the training data set.

NeuPSL Model: The NeuPSL model is a composition of the baseline path-finder and a symbolic component created with NeuPSL that encodes end-points and continuity constraints, i.e., the path from the top left corner of the map to the bottom right corner must be continuous. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a decomposed task structure (Section 4.2.3), with the neural component predicting the cost of each square and the symbolic component predicting shortest valid path.

- **Citeseer and Cora Dataset:** Citeseer and Cora are two widely studied citation network node classification datasets first introduced by Sen et al. (2008). Citeseer consists of 3,327 scientific publications classified into one of 6 topics, while Cora contains 2,708 scientific publications classified into one of 7 topics.

Evaluation: For all experiments, we evaluate models over 5 randomly sampled splits using 20 examples of each topic for training, 200 of the nodes for validation, and 1000 nodes for testing. Prediction performance in this setting is measured by the categorical accuracy of a paper label.

Baseline Model: The baseline neural model for the Citation network settings is a Simple Graph Convolutional Network (SGC) [137]. SGCs are graph convolutional networks with linear activations in the hidden layers to reduce computational complexity. The SGC neural baseline uses bag-of-words feature vectors associated with each paper as node features and citations as bi-directional edges. Then, a MLP is trained to predict the topic label given the SGC-transformed features.

NeuPSL Model: The NeuPSL model is a composition of the baseline SGC and a symbolic component created with NeuPSL that encodes the homophilic structure of the citation network, i.e., two papers connected in the network are more likely to have the same label. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a unified task structure (Section 4.2.3), with both the neural and symbolic components predicting the category a node belongs to.

Target variables indicate the degree to which a paper has a particular topic. The neural classification is used as a prior for the labels of the nodes, and the symbolic component propagates this knowledge to its neighbors.

- **RoadR Dataset:** RoadR is an extension of the ROAD (Road event Awareness Dataset) dataset, initially introduced by Singh et al. (2021). The ROAD dataset was developed to evaluate the situational awareness of autonomous vehicles in various road environments, weather conditions, and times of day. It contains 22 videos, 122k labeled frames, 560k bounding boxes, and a total of 1.7M labels, which include 560k agents, 640k actions, and 499k locations. RoadR builds upon this by adding 243 logical requirements that must be satisfied, further enhancing its utility for testing autonomous vehicles. For instance, a traffic light should never be simultaneously predicted as red and green.

Evaluation: For all experiments, we evaluate models with 15 videos for training and 3 videos for testing. Prediction performance in this setting is measured by the matching boxes using Intersection over Union (IoU) and then multi-class f1. Constraint satisfaction consistency in this setting is the proportion of frame predictions with no constraint violations.

Baseline Model: The baseline neural model for the RoadR dataset is a DETection TRansformer (DETR) model with a ResNet50 backbone [21]. The baseline is trained and applied to detect objects in a frame, along with a multi-label classification for its class labels (e.g., car, red, traffic light, etc.).

NeuPSL Model: The NeuPSL model is a composition of the baseline object detector and classifier and a symbolic component created with NeuPSL that encodes the logical requirements. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a unified task structure (Section 4.2.3), with both the neural and symbolic components predicting the multi-class labels a bounding box belongs to.

- **Logical-Deduction** is a multiple-choice question-answering dataset introduced by [121]. These questions require deducing the order of a sequence of objects given a natural language description and then answering a multiple-choice question about that ordering.

Evaluation: We report results for a single test set of 300 deduction problems, with a prompt containing two examples. Prediction performance in this setting is measured by the accuracy of the predicted multiple-choice answer.

Baseline Model: The baseline neural model for the Logical-Deduction dataset is the models presented in Pan et al. (2023) on GPT-3.5-turbo and GPT-4 [98]. Each model is run using *Standard* and *Chain-of-Thought (CoT)* [133] prompting.

NeuPSL Model: The NeuPSL model is a composition of the baseline LLM that is being prompted to create the constraints within the symbolic program. In all settings, the NeuPSL models create DSPot NeSy-EBMs (Section 3.2.3) in which the neural component creates a symbolic component about the order of objects. The symbolic component then performs constraint satisfaction.

- **Zero-Shot Object Navigation Datasets:** Are datasets in which an embodied agent must navigate to a specific goal object within an unknown environment. For this work, three zero-shot object navigation datasets are considered: MP3D [22],

HM3D [106], and RoboTHOR [40]. MP3D is used in Habitat ObjectNav challenges, containing 2195 validation episodes on 11 validation environments with 21 goal object categories. HM3D is used in the Habitat 2022 ObjectNav challenge, containing 2000 validation episodes on 20 validation environments with 6 goal object categories. RoboTHOR is used in the RoboTHOR 2020 and 2021 ObjectNav challenge, containing 1800 validation episodes on 15 validation environments with 12 goal object categories.

Evaluation: For all experiments, the number of maximum navigation steps is 500. Prediction performance in this setting is measured by the success rate (SR) and the success rate weighted by inverse path length (SPL) [9].

Baseline Model: The baseline neural model for object navigation datasets are CLIP on Wheels (CoW) [54] and ZSON [79]. ZSON uses a CLIP encoder to project the object and image goal into the same embedding space and feed the object goal embedding into an image goal navigation [91] network. CoW uses a gradient-based visualization technique (Grad-CAM [112]) on CLIP to localize goal objects in an egocentric view and a frontier-based exploration technique [139].

NeuPSL Model: The NeuPSL model integrates three core components to guide object navigation: a scene understanding model, a large language model, and a symbolic reasoning layer with NeuPSL rules. The scene understanding model constructs a detailed semantic map, identifying room layouts, objects, and potential frontiers. This map is then passed to the LLM, which infers probable locations for the goal object based on commonsense and context-specific knowledge. Finally, NeuPSL’s symbolic component uses predefined rules to evaluate and select the optimal frontier for navigation, where the target variables represent the specific frontier to approach next. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a decomposed task structure (Section 4.2.3), with the neural component predicting values about commonsense knowledge and symbolic component predicting the frontier it should take.

- **Dialog Structure Induction Datasets:** The dialog structure induction datasets

consist of unlabeled dialog corpora with a latent dialog structure graph representing dialog states and their transition probabilities. Three datasets are used: MultiWoZ 2.1 synthetic [20] and two variations of the Schema-Guided Dialog (SGD) [107] dataset. The SGD datasets include *SGD-synthetic* (with template-generated dialog utterances) and *SGD-real* (in which template-generated utterances are replaced with human paraphrased counterparts). For the SGD-real dataset, three experimental settings are evaluated: *standard generalization* (training and testing within the same domain), *domain generalization* (training and testing across different domains), and *domain adaptation* (training on labeled data from the training domain and unlabeled data from the target domain, with evaluation on target domain data).

Evaluation: The evaluation measures two main aspects: the correctness of the learned latent dialog structure and the quality of the learned hidden representations. Prediction performance is assessed using Adjusted Mutual Information (AMI) [130], while the quality of the hidden representation is evaluated through linear probing, a standard technique for assessing the effectiveness of unsupervised representation learning.

Baseline Model: The baseline neural model for dialog structure induction is the direct discrete variational recurrent neural network (DD-VRNN) [115]. The DD-VRNN builds on the Variational Recurrent Neural Network (VRNN) [26] framework, where a sequence of VAEs [66] is associated with RNN states to model the dialog flow. A key difference with the DD-VRNN is that it directly models the influence of z_{t-1} on z_t , effectively capturing transitions between different latent (dialog) states.

NeuPSL Model: The NeuPSL model is a composition of the baseline DD-VRNN with a symbolic component encoded in NeuPSL, which incorporates a set of dialog constraints. In all settings, the NeuPSL models create DSVar NeSy-EBMs (Section 3.2.1) that use a unified task structure (Section 4.2.3), with both the neural and symbolic components predicting state an utterance is likely to belong to. In the synthetic MultiWoZ setting, 11 structural, domain-agnostic dialog rules are introduced to capture general dialog behaviors, showcasing how even a small set of expert-designed

rules can significantly improve generalization. For the SGD settings, a single dialog rule is added to capture the correlation between dialog acts and utterances containing certain keywords (e.g., utterances with "hello" are likely to belong to the "greet" state). This single rule demonstrates the potential performance gains achievable with minimal prior domain-specific information.

- **Synthetic Mixture of Symbolic Experts Dataset:** The synthetic Mixture of Symbolic Experts dataset is designed to test a model’s ability to reason with both local node features and overarching structural patterns. This dataset consists of disjoint, fully connected vertex and edge subgraphs, with each subgraph conforming to one of two types: 1) nodes with random labels but consistent features, or 2) nodes with a shared label but random features. Experiments are conducted in three inductive settings, where new nodes are added to pre-existing subgraphs, with the following information concatenated to the node features:

- $OH + OH$: One-hot encoding of the label combined with a one-hot encoding of the subgraph ID.
- $G + OH$: A Gaussian sample conditioned on the label, combined with a one-hot encoding of the subgraph ID.
- $G + G$: Gaussian samples conditioned on both the label and the subgraph ID.

Evaluation: The dataset consists of 25 subgraphs, with each subgraph containing between 10 and 15 nodes. The node label space includes four unique labels, and an equal number of subgraphs are generated from each symbolic structure type, with each subgraph containing at least two nodes in the training set. Each experiment is conducted on five random splits using a 60/30/10 train-validation-test partition. Prediction performance is measured by categorical accuracy on node labels.

Baseline Model: The baseline neural models for this dataset are a Multi-Layer Perceptron (MLP) and a Graph Neural Network (GNN). The MLP consists of an input layer, a single hidden layer, and an output layer. The GNN follows the GraphSAGE framework proposed by [61].

NeuPSL Model: The NeuPSL model integrates the baseline MLP with a symbolic component implemented using NeuPSL, encoding constraints for label propagation, and local feature consistency. In all settings, the NeuPSL models create DSPar NeSy-EBMs (Section 3.3) that use a decomposed task structure (Section 4.2.3), with both the neural component predicting the weight of each constraint and the symbolic component predicting class a node belongs to.

7.2 NeSy Inference and Learning

This section begins the empirical evaluation by investigating NeuPSL’s generality and versatility as a NeSy system, showcasing its applicability across a wide range of real-world problems and illustrating when NeSy systems provide significant benefits. Rather than directly comparing NeuPSL to other NeSy approaches, the analysis emphasizes the distinct advantages that NeuPSL—and by extension, NeSy-EBMs (Section 3.1)—offer over purely neural or purely symbolic methods in specific tasks. These experiments primarily address RQ1, which explores how effectively NeSy methods can enhance model accuracy and reasoning capabilities. Additionally, this section contributes to RQ4 by examining how symbolic prior knowledge influences neural representations and to RQ5 by assessing the impact of fine-tuning subsymbolic methods with symbolic information to improve generalization.

7.2.1 Inference

Let’s begin the experimental evaluation by investigating the benefits of employing NeSy methods for inference tasks, specifically focusing on constraint satisfaction, joint reasoning, and few/zero-shot reasoning. In all experiments within this subsection, a modular training approach (Section 4.3.3.1) is applied to independently train the neural and symbolic components, ensuring that inference operates within a well-defined solution space. In this setup, neural components are fully supervised and trained on the entire training dataset, while symbolic weights are optimized through random grid search. For instance, in tasks like MNIST-Add and Visual Sudoku, the neural models are pre-trained on MNIST

digit labels. In the subsequent subsection of learning experiments (Section 7.2.1), supervision will be restricted to the overall task, e.g., only providing labels over for the addition sum. Following this modular training phase, NeuPSL inference is employed to make sure the neural component predictions are consistent with domain knowledge and logical constraints. For these inference tasks, a *DSVar* modeling paradigm (Section 3.2.1), representing the *neural as symbolic variable* architectural axiom (Section 2.3.4), is applied to datasets such as MNIST-Add-k, Visual Sudoku, Pathfinding, RoadR, Citeseer, and Cora. Additionally, a *DSPot* modeling paradigm, representing the *sampling neural for symbolic* architectural axiom (Section 2.3.2), is employed for tasks like Logical Deduction.

Constraint Satisfaction: To study constraint satisfaction, the dataset configurations outlined in Section 7.1 for Visual-Sudoku, Pathfinding, and RoadR. Additionally, consider the following variant of the MNIST-Add- k dataset:

- **MNIST-Add k :** The $k = 1, 2, 4$ MNIST-Add k datasets with the sums of the MNIST-Add- k equations available as observations during inference. Prediction performance is measured by the accuracy of the image classifications.

The MNIST-Add- k variant incorporates the known sum during inference, allowing the model to adjust the predictions from the neural component based on logical consistency. For instance, consider an MNIST-Add-1 example with the equation $[\mathbf{3}] + [\mathbf{5}] = \mathbf{8}$. If the neural component misclassifies the first MNIST digit, $\mathbf{3}$, as an 8 with low confidence, but correctly classifies the second MNIST digit, $\mathbf{5}$, as a 5 with high confidence, the model can use the observed sum, 8, to guide the correction of the first digit label.

In Table 7.1, Table 7.2, and Table 7.3, the prediction performance and constraint satisfaction consistency of a neural baseline model and the NeuPSL model are reported across the MNIST-Add k , Visual-Sudoku, Pathfinding, and RoadR datasets. Across all experimental settings, the baseline neural models frequently violate constraints within the test datasets, particularly as the complexity of the constraints increases. This pattern is most pronounced in the MNIST-Add k datasets, where the consistency of constraint satisfaction decreases significantly as the number of digits, k , increases. The primary

Table 7.1: Digit accuracy and constraint satisfaction consistency of the ResNet18 and NeuPSL models on the MNIST-Add- k and Visual-Sudoku datasets.

	ResNet18		NeuPSL	
	Digit	Acc. Consistency	Digit	Acc. Consistency
<i>MNIST-Add1</i>	97.60 \pm 0.55	93.04 \pm 1.33	99.80 \pm 0.14	100.0 \pm 0.00
<i>MNIST-Add2</i>		86.56 \pm 2.72	99.68 \pm 0.22	100.0 \pm 0.00
<i>MNIST-Add4</i>		75.04 \pm 4.81	99.72 \pm 0.29	100.0 \pm 0.00
<i>Visual-Sudoku</i>		70.20 \pm 2.17	99.37 \pm 0.11	100.0 \pm 0.00

Table 7.2: Accuracy of finding a minimum cost path (Min. Cost Acc.) and consistency in satisfying continuity constraints (Continuity) of the ResNet18 and NeuPSL models on the Pathfinding dataset.

	ResNet18		NeuPSL	
	Min. Cost	Acc. Continuity	Min. Cost	Acc. Continuity
<i>Pathfinding</i>	80.12 \pm 22.44	84.80 \pm 17.11	90.02 \pm 11.70	100.0 \pm 0.00

Table 7.3: Object detection F1 and constraint satisfaction consistency of the DETR and NeuPSL models on the RoadR dataset.

	DETR		NeuPSL	
	F1	Consistency	F1	Consistency
<i>RoadR</i>	0.457	27.5	0.461	100.0

reason for this decline is that the baseline ResNet18 model treats each digit prediction independently. Similarly, in the RoadR dataset, the DETR baseline adheres to road event constraints only 27.5% of the time, illustrating a clear limitation in handling structured dependencies in real-world scenarios.

In contrast, the NeuPSL models consistently satisfy all constraints across all datasets, achieving 100% consistency (RQ1). In short, these models use an optimization process to guarantee that all constraints are satisfied, with the trade-off of changing the neural predictions and runtime costs. Prediction performance gains from constraint

satisfaction are possible when the neural component accurately quantifies its confidence. The symbolic component uses the confidence of the neural component and the constraints together to correct the neural model’s erroneous predictions. For example, if the neural model misclassifies a digit in the MNIST-Add k dataset but assigns a low confidence score to this prediction, NeuPSL can use the sum constraint to adjust the prediction in the hope that the confident prediction is correct.

Joint Reasoning: Unlike MNIST-Add k , Visual-Sudoku, Pathfinding, and RoadR, which involve enforcing hard constraints on the target variables, the citation network datasets—Citeseer and Cora—demonstrate the ability of NeSy methods to handle joint reasoning with soft constraints on a supervised dataset. For these citation networks, NeuPSL enhances prediction accuracy by leveraging the homophilic structure, where linked papers are more likely to share topic labels. Moreover, the symbolic relations can be defined over a general and potentially large number of nodes in the network, i.e., a node can be connected to any number of neighbors.

Table 7.4: Node classification accuracy of the SGC and NeuPSL models on the Citeseer and Cora datasets.

		SGC		NeuPSL
<i>Citeseer</i>		65.14 \pm 2.96		66.52 \pm 3.26
<i>Cora</i>		80.90 \pm 1.54		81.82 \pm 1.73

Table 7.4 presents the prediction performance for the baseline neural model and the NeuPSL model on the citation network node classification tasks. In both Citeseer and Cora, NeuPSL consistently outperforms the baseline, with statistically significant improvements confirmed by a paired t-test (p-values < 0.05) (RQ1). It is crucial to highlight that NeuPSL achieved these gains without additional learning; instead, after each node’s label was predicted, an optimization process was applied to smooth predictions based on neighboring nodes (RQ5).

Few/Zero-Shot Reasoning: The final set of inference experiments explores NeuPSL’s joint reasoning capabilities in scenarios where the neural model has been provided with zero-shot or few-shot information for reasoning. In the zero-shot object navigation setting, a large language model (LLM) is queried to supply commonsense knowledge, which is then integrated into NeuPSL’s constraints to influence the frontier selection process. At a high level, this approach aims to prioritize frontiers closer to objects where the goal object is likely to be found or in rooms where the goal object is expected to be located. Similarly, in the logical deduction question-answering task, NeuPSL leverages an LLM to generate symbolic rules based on dependencies inferred from natural language.

Table 7.5: Comparison of accuracy in answering logical deduction questions using two large language models, GPT-3.5-turbo and GPT-4 [98], across three methods: Standard, Chain of Thought (CoT), and NeuPSL.

	LLM	Standard	CoT	NeuPSL
<i>Logical Deduction</i>	<i>GPT-3.5-turbo</i>	40.00	42.33	70.33
	<i>GPT-4</i>	71.33	75.25	90.67

Table 7.5 and Table 7.6 present the prediction performance of baseline models compared to NeuPSL on the logical deduction and zero-shot object navigation tasks, respectively. In the logical deduction task, NeuPSL demonstrates a substantial 15% im-

Table 7.6: Comparison of success rate (SR) and success rate weighted by inverse path length (SPL) in zero-shot object navigation on MP3D [22], HM3D [106], and RoboTHOR [40] benchmarks, across three methods: CLIP on Wheels (CoW) [54], ZSON [79], and NeuPSL (ESC) [145]

	ZSON		CoW		NeuPSL (ESC)	
	SPL↑	SPL↑	SR↑	SPL↑	SR↑	SPL↑
<i>MP3D</i>	4.8	15.3	6.3	11.1	17.7	36.1
<i>HM3D</i>	12.6	25.5	-	-	25.2	44.0
<i>RoboTHOR</i>	-	-	10.0	16.3	22.2	38.1

provement over the LLM alone (RQ1). This improvement is particularly notable given the occasional generation of syntactically invalid or logically infeasible constraints by the LLM. Specifically, the LLM successfully produced valid rules 89.0% of the time with GPT-3.5-turbo and 98.7% with GPT-4. Future work could enhance this setup by incorporating self-refinement techniques, similar to those proposed in [99], to improve the feasibility and accuracy of rule generation, further boosting NeuPSL’s reasoning capabilities. In the object navigation task, NeuPSL achieves the highest performance across both SR and SPL metrics (RQ1). Notably, on the MP3D dataset, NeuPSL outperforms CoW with a 269% improvement in SPL and a 225% improvement in SR. Additionally, NeuPSL shows strong robustness across multiple datasets without requiring extensive pre-training, unlike ZSON, which is specifically trained on HM3D for image-goal navigation but suffers from performance degradation when applied to new environments like MP3D. NeuPSL’s consistent results across all datasets underscore its generalizability and resilience to domain shifts, highlighting its potential for broad application in varied environments (RQ5).

7.2.2 Learning

Having explored the advantages of NeuPSL and NeSy methods in inference tasks, this section shifts focus to investigate their effectiveness in learning scenarios. Unlike the modular training approach utilized earlier, all experiments in this subsection adopt a joint training strategy, where both neural and symbolic parameters are optimized concurrently. For further details on the specific learning techniques employed, please refer to the work of Dickens (2024). The experiments are conducted across three distinct learning settings:

- **Distant Supervision Learning:** This setting uses a decomposed task structure (Section 4.2.3) for distant supervision learning (Section 4.4.1) with full supervision provided for the higher-level task, while the lower-level task receives no direct supervision.
- **Semi-Supervised Learning:** This experiment presents two semi-supervised settings: (1) Distant supervision (Section 4.2.3) semi-supervised learning where the NeSy method is designed with decomposed task structure (Section 4.2.3). (2) Structure-

informed semi-supervised learning where the NeSy method is designed with a unified task structure (Section 4.2.3).

- **Zero-Shot Learning:** In this setting, no direct supervision is provided at any level. The model relies entirely on structure-informed learning (Section 4.4.2) to guide its predictions.

For all experiments in these learning settings, the *DSVar* modeling paradigm (Section 3.2.1) is employed across the dataset settings. This paradigm aligns with the *neural as symbolic variable* architectural axiom (Section 2.3.4), enabling the integration of neural and symbolic reasoning for collaborative learning.

Distant Supervision Learning: Let’s begin by examining the decomposed task distant supervision learning experiments. In these experiments, the neural component will be provided with no labels, and instead, the representation for the lower tasks must be learned through the gradient passed back from the symbolic component. For example, in MNIST-Add, the individual MNIST images do not have labels but instead have the resulting sum of two MNIST numbers as a label. This setup is explored across different sample sizes, with MNIST-Add1 evaluated on 300, 3,000, and 25,000 examples, and MNIST-Add2 on 150, 1,500, and 12,500 examples. A direct gradient decent learning algorithm (Section 4.3.3.2) is applied to train each setting.

Table 7.7: Test set accuracy and standard deviation on MNIST-Add experiments. Results reported here are run and averaged over ten splits.

Method	MNIST-Add1			MNIST-Add2		
	300	3,000	Number of Additions 25,000	150	1,500	12,500
CNN	17.16 \pm 00.62	78.99 \pm 01.14	96.30 \pm 00.30	01.31 \pm 00.23	01.69 \pm 00.27	23.88 \pm 04.32
NeuPSL	82.58 \pm 02.56	93.66 \pm 00.32	97.34 \pm 00.26	56.94 \pm 06.33	87.05 \pm 01.48	93.91 \pm 00.37

Table 7.7 presents the prediction performance of baseline models compared to NeuPSL on the MNIST-Add1 and MNIST-Add2 data settings. In all cases, NeuPSL achieves a substantial boost in performance over the baseline neural CNN (RQ1). This experiment

underscores a key advantage of NeSy methods: they effectively reduce the complexity of the problem. In a purely neural setting, the model must learn both digit recognition and the concept of addition, requiring a wide variety of digit combinations to generalize correctly. In contrast, NeuPSL enables a decomposed task structure (Section 4.2.3) approach such that the neural component only needs to learn digit representations, drastically reducing the amount of data to generalize (RQ5). Furthermore, after training, the NeSy approach ensures the correct addition of the recognized digits, a significant advantage over the purely neural method. While a CNN might learn accurate digit representations, it could still fail to combine these predictions correctly to produce an accurate sum.

Semi-Supervised Learning: This subsection explores semi-supervised experiments, where either the lower-level or higher-level task receives partial supervision through labeled data. Specifically, it compares the prediction performance of a neural baseline trained exclusively with a supervised neural loss against a NeuPSL model’s neural component (using the same architecture) trained end-to-end with a NeSy-EBM loss. In both cases, only a subset of the training set labels is provided to the neural component. The experiments examine two distinct semi-supervised settings:

1. **Distant Supervision Semi-Supervised Learning:** In this setting, the NeSy method is designed with a decomposed task structure (Section 4.2.3) for distant supervision learning (Section 4.4.1) where the neural component is partially supervised, and the symbolic component is fully supervised. This setup is used in the MNIST-Add k and visual sudoku dataset experiments.
2. **Structure Informed Semi-Supervised Learning:** In this setting, the NeSy method is designed with a unified task structure (Section 4.2.3) for structure-informed learning (Section 4.4.2) where both the neural and symbolic components are partially supervised, sharing the same labeled data. Furthermore, this setting studies learning with a constraint loss (Section 4.4.3) approach on the dialog structure induction experiments.

For additional semi-supervised experiments on Pathfinding, Citeseer, and Cora datasets,

I direct readers to my colleague’s work [43]. The experiments are conducted using the following dataset variants:

- **MNIST-Add k :** The experiments are conducted on MNIST-Add k datasets for $k = 1, 2$, with the proportion of image class labels in the training data varying over $\{1.0, 0.5, 0.1, 0.05\}$. Prediction performance is evaluated based on the accuracy of the image classifications.
- **Visual-Sudoku:** The Visual-Sudoku dataset explores the semi-supervised setting where the proportion of image class labels in the training data also varies over $\{1.0, 0.5, 0.1, 0.05\}$.
- **Dialog Structure Induction** Three constrained few-shot settings are examined: 1-shot, proportional 1-shot, and 3-shot. In the 1-shot and 3-shot settings, one and three labels per class are provided, respectively. In the proportional 1-shot setting, the number of labels matches the 1-shot setting but is distributed proportionally to class sizes (with no labels provided for classes comprising less than 1% of the data).

A bilevel learning algorithm (Section 4.3.3.3) is applied to train the MNIST-Add k , Citeseer, and Cora datasets. A direct gradient decent learning algorithm (Section 4.3.3.2) is applied to train the dialog structure induction dataset.

Figure 7.1: Average AMI for MultiWoZ, SGD Synthetic, and SGD Real (Standard Generalization, Domain Generalization, and Domain Adaptation) on three constrained few-shot settings: 1-shot, proportional 1-shot, and 3-shot.

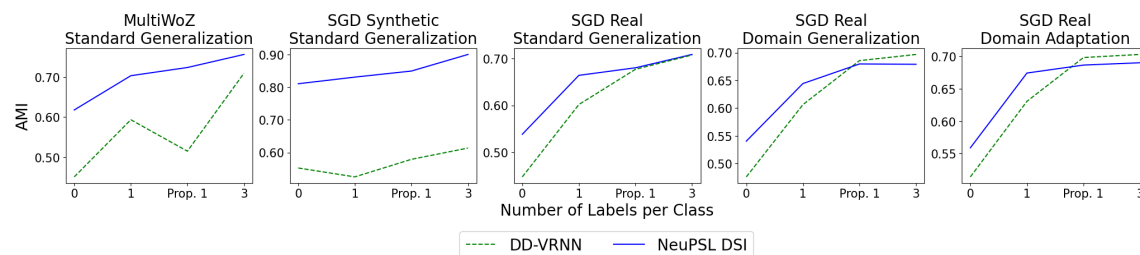


Table 7.8: Digit accuracy of the ResNet18 models trained with varying levels of supervision.

		ResNet18	
		Supervised	NeuPSL Semi-Supervised
<i>MNIST-Add1</i>	Labeled		
	1.00	97.84 ± 0.23	97.40 ± 0.51
	0.50	97.42 ± 0.30	97.02 ± 0.65
	0.10	93.05 ± 0.69	96.78 ± 0.80
	0.05	75.35 ± 0.33	96.82 ± 0.72
<i>MNIST-Add2</i>	Labeled		
	1.00	97.84 ± 0.23	97.22 ± 0.19
	0.50	97.42 ± 0.30	96.84 ± 0.42
	0.10	93.05 ± 0.69	95.14 ± 1.21
	0.05	75.35 ± 0.33	95.90 ± 0.43
<i>Visual-Sudoku</i>	Labeled		
	1.00	97.84 ± 0.23	97.89 ± 0.15
	0.50	97.42 ± 0.30	97.26 ± 0.70
	0.10	93.05 ± 0.69	96.82 ± 0.32
	0.05	75.35 ± 0.33	96.49 ± 0.67

Table 7.8 and Figure 7.1 report the average and standard deviation of the prediction performance for a supervised baseline and a semi-supervised neural component on the MNIST-Add k , Visual-Sudoku, and Dialog Structure Induction datasets. Across these datasets, as the proportion of unlabeled data increases, the semi-supervised NeuPSL neural component consistently outperforms the purely supervised baseline (RQ1). This trend highlights the ability of neural components to leverage either a decomposed task structure or a structure-informed setting within a NeSy system. The benefit of utilizing symbolic knowledge is most pronounced in the lowest supervision settings. For instance, the NeuPSL semi-supervised ResNet18 model achieves over a 20-percentage-point improvement when only 5% of the training labels are available in the MNIST-Add k and Visual-Sudoku datasets. This substantial gain suggests that, in low-supervision scenarios, integrating symbolic domain knowledge into the learning process can significantly enhance model performance. In the Dialog Structure Induction dataset setting, it is clear that the prior knowledge, which, unlike MNIST-Add and Visual Sudoku, is not always correct, and as the number of clean labels increases, the model overfits the symbolic constraints (RQ5).

Therefore, an interesting direction for future work would be enabling models to perform weight learning consistently, where the model adaptively weights the importance of symbolic rules as stronger evidence is introduced.

7.2.3 Zero-Shot Learning

This final learning experiment focuses on zero-shot learning, where neither the neural nor the symbolic components receive any direct supervision. Specifically, this section compares the prediction performance of a neural baseline, trained exclusively with an unsupervised neural loss, against a NeuPSL model’s neural component (using the same architecture) trained end-to-end in a *learning with constraint loss* setting (Section 4.4.3). In this zero-shot scenario, the NeSy method employs a *structure-informed learning* approach (Section 4.4.2) within a *unified task structure* (Section 4.2.3). The experiments in this subsection study the dialog structure induction data settings.

Dataset	Setting	Method	Hidden Representation Learning		Structure Induction (AMI)
			Full (Class-Balanced Accuracy)	Few-Shot (Class-Balanced Accuracy)	
MultiWoZ	Standard Generalization	DD-VRNN	0.804 ± 0.037	0.643 ± 0.038	0.451 ± 0.042
		NeuPSL	0.806 ± 0.051	0.689 ± 0.038	0.618 ± 0.028
SGD Synthetic	Standard Generalization	DD-VRNN	0.949 ± 0.005	0.598 ± 0.019	0.553 ± 0.017
		NeuPSL	0.941 ± 0.009	0.765 ± 0.012	0.826 ± 0.006
SGD Real	Standard Generalization	DD-VRNN	0.661 ± 0.015	0.357 ± 0.015	0.448 ± 0.019
		NeuPSL	0.663 ± 0.015	0.517 ± 0.021	0.539 ± 0.048
	Domain Generalization	DD-VRNN	0.268 ± 0.012	0.320 ± 0.029	0.476 ± 0.029
		NeuPSL	0.299 ± 0.009	0.528 ± 0.026	0.541 ± 0.036
	Domain Adaptation	DD-VRNN	0.308 ± 0.011	0.505 ± 0.015	0.514 ± 0.028
		NeuPSL	0.297 ± 0.025	0.541 ± 0.023	0.559 ± 0.045

Table 7.9: Test set performance on all datasets. All reported results are averaged over 10 splits. The highest-performing methods per dataset and learning setting are bolded. A random baseline has AMI zero and class-balanced accuracy equal to inverse class size.

Table 7.9 presents the results of NeuPSL and the purely data-driven *DD-VRNN* in strictly unsupervised settings. NeuPSL outperforms the DD-VRNN on AMI by 4%-27%, depending on the specific dataset and experimental setting, and achieves this improvement while maintaining or even enhancing the quality of the hidden representation (RQ1). This indicates that the symbolic constraints not only improve prediction performance but also

lead to more robust and structured internal representations (RQ4). An interesting observation arises from comparing AMI performance on the SGD-real dataset across different generalization settings (standard generalization versus domain generalization/adaptation). NeuPSL consistently outperforms DD-VRNN in each setting, though the advantage is slightly reduced in the more challenging domain generalization and adaptation cases. This highlights the potential of NeSy methods as a practical solution when labeled data is scarce or costly to obtain. Such unsupervised training can lay a solid foundation, creating a structured representation that can be further refined if labeled data becomes available (RQ5).

7.3 Comparing NeSy Approaches

In this second collection of experiments, let’s turn our focus the study of prominent neural-symbolic methods. Unlike the previous section, which focused on the advantages of NeSy approaches over purely neural or symbolic methods, this section aims to understand under which scenarios do mainstream NeSy approaches perform optimally (RQ2). The section begins with an overview of the selected NeSy methods, highlighting their distinct modeling paradigms, theoretical foundations, and practical implementations. This introduction is followed by a series of subsections that evaluate each approach within the context of specific datasets, shedding light on their respective strengths and limitations.

7.3.1 Overview of NeSy Models

This subsection offers a concise overview of the NeSy models (outside of NeuPSL) examined in this section. Although the number of approaches studied is limited, these methods represent some of the most mainstream, well-established, and principled approaches in the NeSy field. The NeSy approaches explored in this section, along with their key publications and source code repositories, are summarized below:

DeepProbLog (DPL) [81]: DeepProbLog is a probabilistic neural-symbolic approach incorporating neural predictions as probabilistic distributions over facts within a

probabilistic logic framework. Typical models in DPL are formulated as *deep symbolic parameter* modeling paradigms (Section 3.2.2) using the neural as symbolic parameter architectural type (Section 2.3.3). All DPL results presented here use the model from [81] with default hyperparameters. Code was obtained from <https://github.com/ML-KULEuven/deepproblog>.

DeepStochLog (DSL) [136]: Similar to DeepProbLog, DeepStochLog treats neural predictions as probabilistic distributions over variables or facts within a probabilistic logic framework. However, DSL is based on stochastic definite clause grammars, a form of stochastic logic programming that defines a probability distribution over derivations, allowing for probabilistic reasoning through rule-based structures. Typical models in DSL are formulated as *deep symbolic parameter* modeling paradigms (Section 3.2.2) using the neural as symbolic parameter architectural type (Section 2.3.3). All DSL results use the model from [136] with default hyperparameters. Code was obtained from <https://github.com/ML-KULEuven/deepstochlog>.

Logic Tensor Networks (LTNs) [14]: Logic Tensor Networks are computation graph (Section 4.2.1) fuzzy logic NeSy approaches that incorporate neural predictions in a real-valued soft logic approximation. Typical models in LTNs are formulated as *deep symbolic variable* modeling paradigms (Section 3.2.1) using the neural as symbolic variable architectural type (Section 2.3.4). All LTN results presented here use the model from [14] with default hyperparameters. Code was obtained from <https://github.com/logictensornetworks/logictensornetworks>.

Graph Neural Network (GNN) [61]: Finally, I argue that a Graph Neural Network is considered a NeSy approach, where each node represents the neural network’s predictions, and the symbolic component is the aggregation functions of its neighbors. GNNs are formulated as *deep symbolic variable* modeling paradigms (Section 3.2.1) using the neural as symbolic variable architectural type (Section 2.3.4). All GNN results use an augmented version of GraphSAGE from [61].

Licenses for NeuPSL, DeepProbLog, and DeepStochLog are under Apache License

2.0, and Logic Tensor Networks are under MIT License.

7.3.2 MNIST Addition

This section evaluates and compares the performance of NeuPSL, DeepProbLog, Logic Tensor Networks, and a neural baseline (CNN) on the MNIST-Add task. The primary objective of this experiment is to assess the predictive performance and inherent strengths of three prominent, general-purpose NeSy approaches on a canonical dataset, which employs *distant supervision learning* (Section 4.4.1) within a *decomposed task structure* (Section 4.2.3). By analyzing this setting, I aim to highlight the trade-offs between different NeSy approaches and the neural baseline, offering insights into their respective advantages and limitations. To further examine these trade-offs, the subsequent subsection explores a variant of the MNIST-Add dataset tailored for collective classification, evaluating performance and runtime for inference and learning. For this experiment, ten train splits were generated by randomly selecting, without replacement, $n \in \{600, 6000, 50000\}$ unique MNIST images from the original MNIST train split. Validation and test splits were created similarly, with test splits pulled exclusively from the original MNIST test set ($n = 10000$) to prevent data leakage.

Table 7.10: Test set accuracy and standard deviation on *MNIST-Add*. Results reported here are averaged over the same ten splits. Best results and those within one standard deviation of the best are in bold.

Method	MNIST-Add1				MNIST-Add2	
	300	3,000	Number of Additions		1,500	12,500
			25,000	150		
CNN	17.16 \pm 00.62	78.99 \pm 01.14	96.30 \pm 00.30	01.31 \pm 00.23	01.69 \pm 00.27	23.88 \pm 04.32
LTN	69.23 \pm 15.68	93.90 \pm 00.51	80.54 \pm 23.33	02.02 \pm 00.97	71.79 \pm 27.76	77.54 \pm 35.55
DPL	85.61 \pm 01.28	92.59 \pm 01.40	⁻²	71.37 \pm 03.90	87.44 \pm 02.15	⁻²
NeuPSL	82.58 \pm 02.56	93.66 \pm 00.32	97.34 \pm 00.26	56.94 \pm 06.33	87.05 \pm 01.48	93.91 \pm 00.37

Performance Results: Table 7.10 presents the average accuracy and standard deviation for MNIST-Add1 and MNIST-Add2.¹ The highest average accuracy and results within one standard deviation of the best are highlighted in bold. Several key insights emerge from these results. First and foremost, it is important to highlight that similarly to the inference and learning experiments of NeuPSL, most NeSy methods consistently outperform the baseline across the various data sizes, regardless of their underlying architectural, inference, or learning strategies. While DeepProbLog encountered errors in one setting, it generally shows performance improvements as the number of additions increases. Prior studies have demonstrated that probabilistic logic approaches, such as DeepProbLog, can achieve competitive or better results to NeuPSL in the standard MNIST-Add setting when the full dataset is utilized.

A key difference in performance between the NeSy methods can be traced to how random variables are represented and the underlying inference process. When observing the representation, DeepProbLog has binary random variables, whereas NeuPSL and LTNs adopt a continuous random variables. Notice that in low-data scenarios, such as MNIST-Add1 with 300 additions or MNIST-Add2 with 150 additions, DeepProbLog significantly outperforms both LTN and NeuPSL. Upon further analysis, this is in large part due to a soft logic pitfall (Section 5.3.3): models predicting target variable values around the average. For example, consider the equation $[\mathbf{3}] + [\mathbf{5}] = 8$. Instead of assigning binary representations—e.g., $\mathbf{3} = 3$ with probability 1.0 and $\mathbf{5} = 5$ with probability 1.0—fuzzy logic might assign a softer prediction, such as $\mathbf{3} = 5$ with a low confidence variable assignment, e.g., 0.1. This tendency can be mitigated by introducing more additions into the training dataset (as seen in higher addition settings in Table 7.10) or incorporating a pretraining step (Section 7.4.4) (RQ3).

When comparing the predictive performance of NeuPSL and DeepProbLog (DPL), both optimization-based inference approaches, against Logic Tensor Networks (LTN), a computation graph-based inference approach, a clear disparity emerges: the computation graph-based approach underperforms.² This difference is largely attributable to the ex-

¹In the largest data setting, DeepProbLog produced results that appeared to be random due to an error. Rather than report potentially misleadingly low results, these are indicated with ‘-’.

²Technically, DeepProbLog constructs an SDD [35] which is a computation graph, but this process is

licit reasoning performed by optimization-based methods, which are designed to find the most optimal solution. In contrast, computation graph-based approaches delegate the reasoning process to the neural network, which can result in solution spaces that are harder to optimize, leading to longer convergence times or becoming trapped in local minima. This is particularly evident in the largest MNIST-Add1 setting.

Additionally, optimization-based methods that employ continuous-valued variables and fuzzy logic representations, such as NeuPSL, tend to exhibit lower variance in their results. This behavior stems from the smoothing effect of continuous-valued optimization, which drives predictions toward averaged values. While this can introduce challenges in highly discrete settings, it often leads to more stable and consistent performance across varied data conditions.

Key Takeaways:

1. For discrete problems that can be efficiently reasoned with probabilistic logics, methods like DeepProbLog are advantageous.
2. Fuzzy logic-based approaches, such as NeuPSL and LTNs, achieve similar performance given sufficient data or computational time.
3. Optimization-based methods tend to outperform computation graph-based methods in scenarios requiring complex reasoning, though they may incur higher computational costs, a trade-off explored in subsequent experiments.

7.3.3 MNIST Addion: Overlap

This section extends the comparison of NeuPSL, DeepProbLog, and Logic Tensor Networks through a variation of the MNIST-Add task. The purpose of the following data setting is to explore the collective reasoning capabilities of each NeSy approach and to understand how each approach’s inference and learning times are effected. In short, the variation introduced to the MNIST-Add task involves reusing digits across multiple addition examples, thereby introducing *overlap*. Figure 7.2 illustrates the process of in-
done in an instance-based approach, which is akin to reasoning in this setting (Section 4.2.2).

Figure 7.2: Example of overlapping MNIST images in MNIST-Add1. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros.

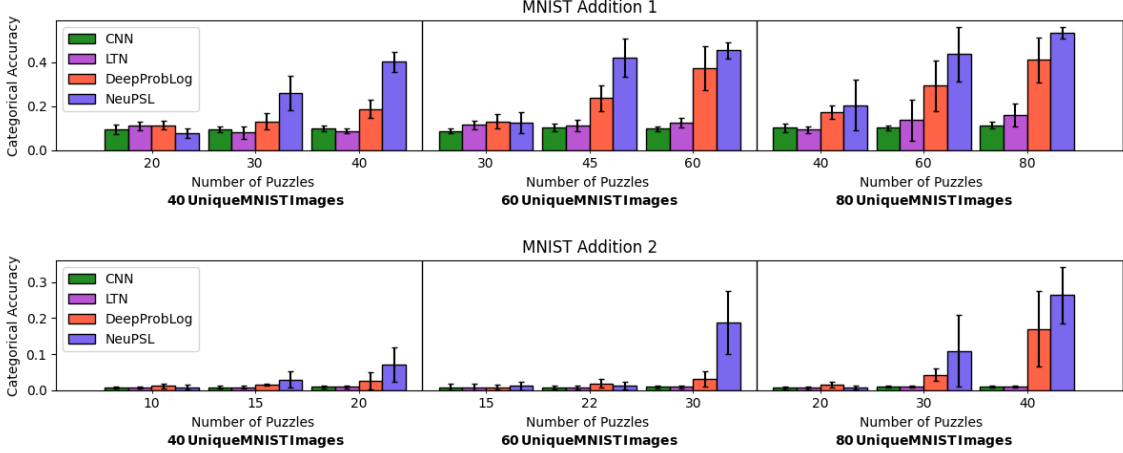
<div style="border: 2px solid red; display: inline-block; padding: 2px;">0</div> + 3 = 3	<div style="border: 2px solid red; display: inline-block; padding: 2px;">0</div> , 3	<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> , 1		<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> + 3 = 3	<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> , 3	<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> , 1
<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> + 1 = 1	$\begin{pmatrix} 0 & 3 \\ 0 & 3 \\ 1 & 2 \\ 2 & 1 \\ 3 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$		<div style="border: 2px solid blue; display: inline-block; padding: 2px;">0</div> + 1 = 1	$\begin{pmatrix} 0 & 3 \\ 0 & 3 \\ 1 & 2 \\ 2 & 1 \\ 3 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$

Introducing overlap and how joint reasoning models narrow the potential label space when MNIST images are reused. For instance, in the scenario depicted in Figure 7.2, the same MNIST image of a zero is used in two separate addition problems. To satisfy both addition constraints, the potential label space for this image is restricted, excluding values such as two or three, as they would violate one of the addition rules. In contrast, models performing independent reasoning lack the capacity to enforce such cross-example constraints, resulting in less consistent predictions.

For this analysis, the focus is placed on low-data settings to evaluate whether the joint reasoning capabilities of NeSy systems can effectively leverage additional structural information to compensate for limited data availability. To introduce overlap, a set of n unique MNIST images is sampled and reused to create $(n + m)/2$ MNIST-Add1 additions and $(n + m)/4$ MNIST-Add2 additions. The degree of overlap is varied by adjusting $m \in \{0, n/2, n\}$, while performance is compared across datasets with $n \in \{40, 60, 80\}$. Results are evaluated over ten test sets, each comprising 1,000 MNIST images, with the degree of overlap proportional to the respective training set.

Performance Results: Figure 7.3 summarizes the average performance across varying overlap settings. Each panel evaluates a fixed number of unique MNIST images while varying the number of additions. For instance, the upper-left panel illustrates results for MNIST-Add1 using 40 unique images to generate 20, 30, and 40 additions. Initially, in scenarios with no overlap, the symbolic inference lacks sufficient structure to accurately identify the correct digit labels for training the neural models. However, as the number of additions increases while keeping the number of unique MNIST images fixed, both DPL

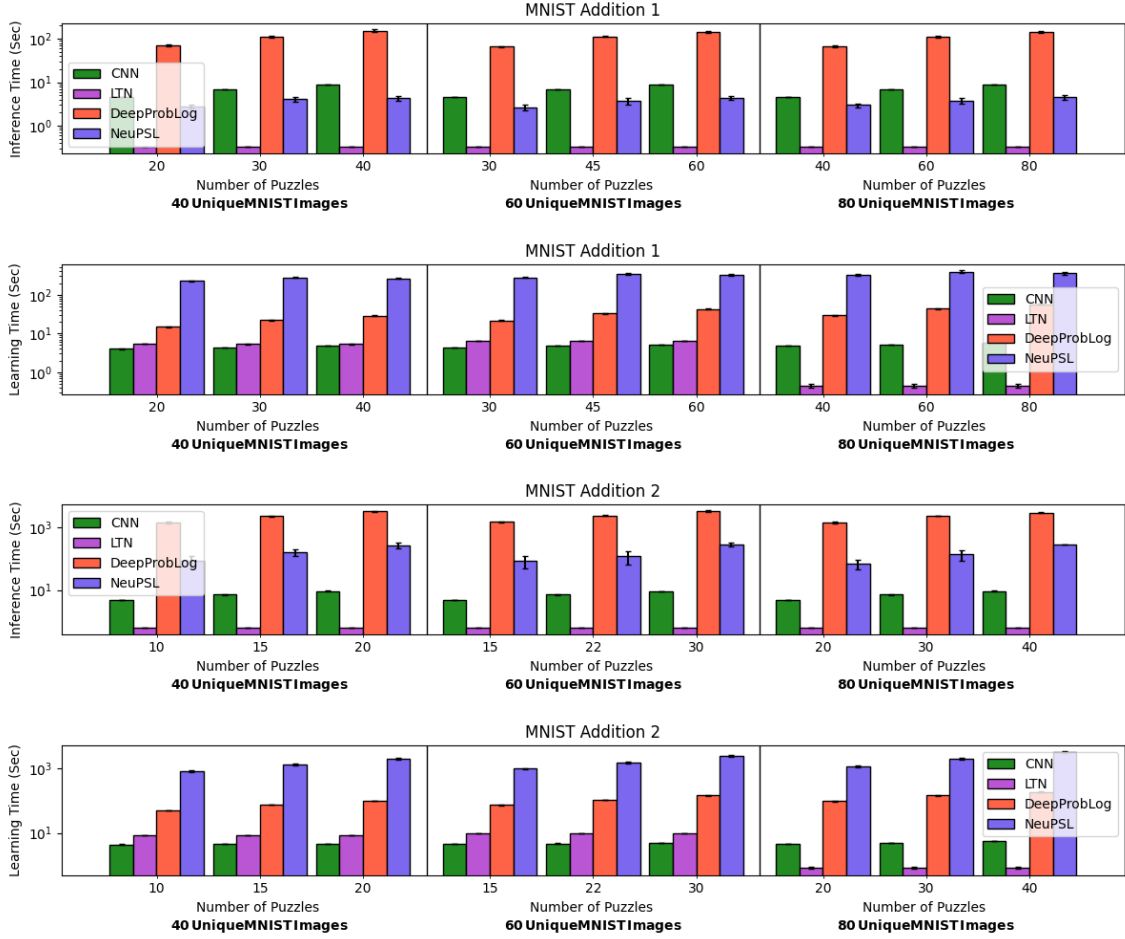
Figure 7.3: Average test set accuracy and standard deviation on *MNIST-Add* datasets with varying amounts of overlap.



and NeuPSL demonstrate significant improvements in prediction performance by effectively leveraging the additional joint information (RQ1). This underscores the strength of NeSy methods in utilizing structural constraints to enhance learning.

A key insight emerges when comparing optimization-based approaches (NeuPSL and DeepProbLog) with computation graph-based methods like LTN. Both optimization-based approaches capitalize on the structural overlap, converging toward stable solutions. Conversely, LTNs fail to benefit from the additional structure introduced by overlap, as they are not designed for collective reasoning across multiple constraints. Among the optimization-based methods, NeuPSL consistently outperforms DeepProbLog due to its ability to solve all equations collectively. While DeepProbLog can theoretically perform similar collective reasoning, its reliance on exact probabilistic computations leads to significant scalability challenges as the structural size grows. Achieving comparable collective reasoning in DeepProbLog would require extensive modifications to its underlying framework, making it less practical for this large-scale problems.

Figure 7.4: Inference and learning time for *MNIST-Add* experiments.



Inference and Learning Runtime Results: To gain further insight into the performance trade-offs of these approaches, Figure 7.4 presents the inference and learning runtimes associated with the overlap experiments. The results reveal distinct differences in how each method balances computational efficiency with performance. For NeSy methods involving complex symbolic inference (DPL and NeuPSL), a trade-off becomes evident. NeuPSL performs inference approximately an order of magnitude faster than DPL but requires longer training times due to its approach of taking full gradient steps over the entire

training dataset (RQ2). In contrast, DPL employs batched stochastic gradient steps over a handful of additions, which reduce training time but result in slower inference. As such, if DeepProbLog were to perform collective inference over all equations, the learning runtime would be slower by orders of magnitude. In the following data setting, it will become apparent that this exact computational cost will be impractical in larger settings. Finally, for the methods that do not involve a complex symbolic inference (CNN and LTNs), the inference and learning times are all considerably faster, unsurprisingly.

Key Takeaways:

1. Collective reasoning can overcome data scarcity with the addition of a more beneficial structure (RQ1).
2. DeepProbLog can leverage the collective structure, but due to the inherent computational cost in the subprocess of inference, it will limit its applicability (RQ2).
3. NeuPSL can most effectively handle collective reasoning due in part to its ability to perform fast exact MAP inference over a relaxed problem (RQ2).

7.3.4 Citeseer and Cora

This section extends the comparison between NeuPSL and DeepProbLog to provide a deeper understanding of the computational trade-offs between these two methods by implementing them on two well-studied graph datasets: Citeseer and Cora. Additionally, the experiments evaluate two NeSy approaches that are particularly suited for larger graph-based problems: DeepStochLog [136] and Graph Convolutional Networks (GCN) [67].³ Furthermore, two baselines are included in the evaluation to isolate the contributions of the distinct neural and symbolic components used in the NeuPSL model: LP_{PSL} , which represents a purely symbolic reasoning approach, and $Neural_{PSL}$, which focuses solely on neural reasoning without symbolic integration. The experiments are conducted

³There is some debate as to whether GCNs qualify as NeSy systems. Fundamentally, their symbolic component can be viewed as an aggregation function over neighboring nodes. This aligns closely with the Symbolic as Neural architectural choice.

using ten randomly sampled splits, with 5% of the nodes designated for training, 5% for validation, and 1,000 nodes for testing. Results are averaged across these splits to provide a robust evaluation of performance, allowing for a comprehensive comparison of how different NeSy approaches handle the computational and structural challenges inherent in graph-based reasoning tasks.

Table 7.11: Test set accuracy and inference runtime in seconds on two citation network datasets.

Method	Citeseer		Cora	
	(Accuracy)	(Seconds)	(Accuracy)	(Seconds)
Neural _{PSL}	57.76 ± 1.71	-	57.12 ± 2.13	-
LP _{PSL}	50.88 ± 1.18	-	73.32 ± 2.39	-
DeepProbLog	timeout	timeout	timeout	timeout
DeepStochLog	61.30 ± 1.44	34.42 ± 0.87	69.96 ± 1.47	165.28 ± 4.49
GCN	67.50 ± 0.57	3.10 ± 0.04	79.52 ± 1.13	1.31 ± 0.01
NeuPSL	68.48 ± 1.22	4.23 ± 0.05	81.22 ± 0.79	4.07 ± 0.14

Performance Results: Table 7.11 summarizes the average prediction performance on the Citeseer and Cora datasets. Consistent with previous experiments, all NeSy approaches demonstrate significant improvements in prediction performance by effectively leveraging joint structural information (RQ1). Among the methods that did not timeout, both GCN and NeuPSL show superior performance compared to the probabilistic logic approaches, DeepStochLog and DeepProbLog (RQ2).

Notably, NeuPSL outperforms GCN despite its slightly lower representational capacity. This result can be attributed to how NeuPSL constructs its underlying symbolic component. Specifically, NeuPSL ties the parameter associated with each soft constraint to the original templated rule, enforcing universal weights for all connections. While this approach simplifies the parameterization and allows for efficient reasoning, it limits the model’s ability to capture heterogeneous propagation behaviors within portions of a class. In contrast, GCN’s architecture can represent these variations more flexibly. This structure will be further examined in the next experiment, which explicitly studies the effects of

parameter sharing.

Inference and Learning Runtime Results: In addition to predictive performance, Table 7.11 reports the inference and learning runtimes for the graph datasets. Similar to the MNIST-Overlap experiments, NeuPSL’s scalable collective inference enables runtimes comparable to those of a feed-forward neural model. In contrast, DeepProbLog was unable to complete the experiment within the allotted time, while its scalable counterpart, DeepStochLog, required significantly longer runtimes than both GCN and NeuPSL. NeuPSL’s ability to efficiently scale its joint inference process becomes evident in this setting, achieving higher accuracy while maintaining substantial runtime advantages. Specifically, NeuPSL achieves an 8x and 40x speed-up over DeepStochLog on the Citeseer and Cora datasets, respectively (RQ2). These results highlight NeuPSL’s strength in handling large-scale inference problems through its ability to perform fast, exact MAP inference over relaxed symbolic representations.

Key Takeaways:

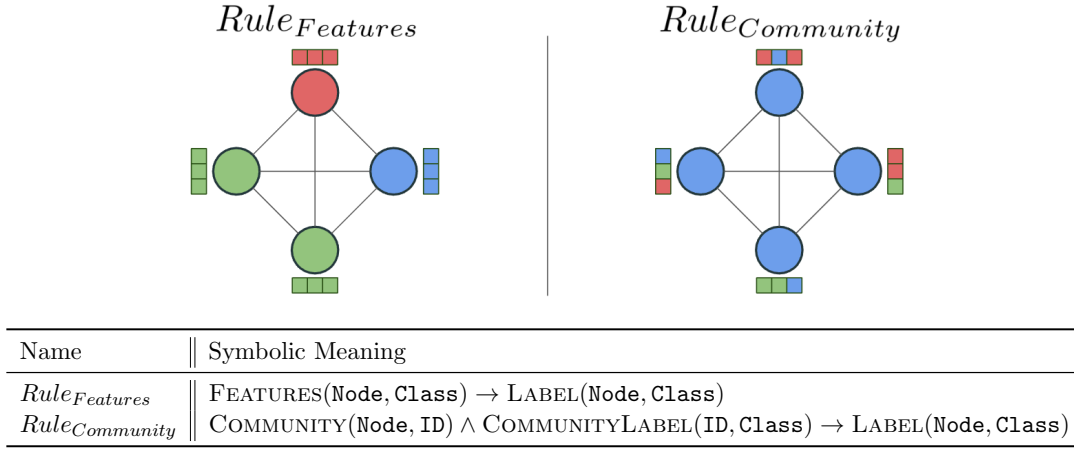
1. Probabilistic logic-based NeSy approaches are capable of exact inference on discrete problems; however, they face significant runtime trade-offs, which constrain their scalability for large-scale reasoning tasks. Alternatively, these methods can opt for inexact inference [80] to improve efficiency, but this comes at the cost of increased variance and reduced predictive performance.
2. NeuPSL demonstrates its scalability and efficiency by leveraging relaxed MAP inference, enabling it to handle larger inference problems while maintaining a compact parameter space and delivering competitive performance (RQ2).

7.3.5 Synthetic Mixture of Symbolic Experts

This section extends the comparison between NeuPSL and Graph Convolutional Networks (GCNs) to provide a deeper understanding of the trade-offs associated with parameter sharing and the ability of these methods to handle noisy data. The evaluation

is conducted on a synthetic mixture of symbolic experts dataset, designed to challenge systems to reason effectively with both local features and global structure. Two variations of NeuPSL are introduced for this experiment: $NeuPSL_{shared}$, which uses globally shared parameters learned through a symbolic learning algorithm (e.g., gradient descent), and $NeuPSL_{individual}$, which employs a neural network to predict the weights of individual grounded rules. This distinction highlights scenarios where parameter sharing might struggle to capture diverse or noisy structures, making $NeuPSL_{individual}$ a more flexible alternative.

Figure 7.5: Rules, symbolic meaning, and graphical representation used to generate features and labels for the synthetic datasets.



In short, the synthetic dataset is defined as a set of disjoint vertex and edge communities that are fully connected. Each community adheres to an underlying rule governing its node’s labels and features. Figure 7.5 summarizes the underlying rules, their symbolic interpretations, and a graphical example of each. Communities generated according to $Rule_{Features}$ will have random node labels but features directly correlating to these labels. On the other hand, communities generated based on $Rule_{Community}$ will have a single common label, but node features are not correlated with the community label. In all experiments the number of communities is $k = 25$, and the node label space is $L = \{0, 1, 2, 3\}$. The minimum and maximum community sizes were set to $a_{min} = 10$ and

$a_{max} = 15$, respectively. It was ensured that an equal number of communities generated from $Rule_{Features}$ and $Rule_{Community}$ were present. Each experiment was performed on 5 splits using 60/30/10 train-test-valid partitions of the inductive setting. Every community was generated to contain at least two nodes within the train set.

Performance Results (Parameter Sharing): To begin, let’s assess the ability of each model to reason about the problem when the features for communities generated from $Rule_{Features}$ directly represent the underlying labels. This direct representation is evaluated using the $OH + OH$ setting and the $G + OH$ and $G + G$ settings when the covariance identity matrix is multiplied by a small scalar value of 0.1. In this setting, most node labels are identifiable from the features and graph structure.

Table 7.12: Average categorical accuracy on the highest correlation between features and labels for $OH + OH$, $G + OH$, and $G + G$ data settings. Best-performing methods are in bold.

Method	$OH + OH$	$G + OH$	$G + G$
MLP	99.09 ± 1.82	88.10 ± 5.69	89.04 ± 2.72
GNN	98.43 ± 1.94	94.56 ± 1.98	81.75 ± 2.65
$NeuPSL_{shared}$	81.07 ± 5.14	87.06 ± 5.61	86.71 ± 5.46
$NeuPSL_{individual}$	100.00 ± 0.00	100.00 ± 0.00	93.35 ± 2.23

Table 7.12 presents the average categorical accuracy and standard deviation for each model in the first experimental setting. Across all cases, $NeuPSL_{individual}$ consistently outperforms the other approaches, with only a minor drop in performance on the most challenging features ($G + G$). Interestingly, $NeuPSL_{shared}$ struggles to simultaneously model the symbolic structure of the two distinct node community types, highlighting the limitation of shared parameters for structure. This difference underscores the flexibility of $NeuPSL_{individual}$, which allows the neural model to adaptively predict rule weights, enabling it to handle heterogeneous community structures more effectively. Furthermore, while GNNs demonstrate generalization capabilities, they lack explicit rule encodings and consequently make errors that $NeuPSL_{individual}$ can avoid (RQ2). In essence,

$NeuPSL_{individual}$ leverages constraints to guarantee predictions and relies on the neural model to determine the most applicable constraints. This stands in contrast to GCNs, which lack such guarantees, as they rely solely on feature aggregation without explicit structural reasoning.

Performance Results (Noise): Now, let’s shift our focus to evaluating how each model performs as the correlation between the features of communities generated under $Rule_{Features}$ and their node labels diminishes. In essence, this introduces increasing levels of noise into the problem, simulating real-world scenarios where feature-label relationships are imperfect or noisy. This degradation in correlation is modeled by progressively increasing the covariance of the features for the $(G + OH)$ and $G + G$ settings. The covariance values used in this experiment are drawn from the range $\{0.1, 1.0, 10.0, 50.0, 100.0\}$, with higher covariance values representing greater variability and less structured alignment between features and labels.

Table 7.13: Average categorical accuracy on varying covariance matrices used for synthetic data generation in the $G + OH$ and $G + G$ data settings. Higher covariance results in a lower correlation between features and labels. Best-performing methods are in bold.

Features	Covariance	MLP	GNN	$NeuPSL_{shared}$	$NeuPSL_{individual}$
$G + OH$	0.1	88.10 ± 5.69	94.56 ± 1.98	87.06 ± 5.61	100.00 ± 0.00
	1.0	88.59 ± 5.87	95.91 ± 2.35	87.06 ± 5.61	100.00 ± 0.00
	10.0	81.71 ± 7.31	90.91 ± 4.38	85.23 ± 5.25	96.35 ± 2.23
	50.0	76.92 ± 5.63	71.30 ± 7.96	76.11 ± 3.21	78.85 ± 6.27
	100.0	69.70 ± 3.22	60.53 ± 8.58	69.47 ± 5.06	62.44 ± 5.70
$G + G$	0.1	89.04 ± 2.72	81.75 ± 2.65	86.71 ± 5.46	93.36 ± 5.06
	1.0	89.27 ± 2.44	82.42 ± 6.40	86.71 ± 5.46	92.48 ± 3.99
	10.0	78.84 ± 1.71	80.15 ± 3.17	84.44 ± 4.69	91.36 ± 2.61
	50.0	53.00 ± 4.08	58.30 ± 3.37	74.18 ± 4.91	72.57 ± 6.58
	100.0	42.10 ± 4.63	54.14 ± 6.19	69.39 ± 5.09	51.66 ± 6.89

Table 7.13 presents the average and standard deviation of categorical accuracy achieved by each model across varying levels of noise in the data. As expected, increasing covariance, which reduces the representativeness of features relative to the underlying

labels, negatively impacts the performance of all models. Notably, the underlying graph structure, including edges and observed node labels, remains consistent across all settings. This consistency allows methods that leverage symbolic label propagation, such as $NeuPSL_{shared}$ and $NeuPSL_{individual}$, to maintain relatively accurate predictions for approximately half of the nodes. This is because half of the communities are generated from $Rule_{Community}$, where the labels are entirely governed by symbolic relationships rather than noisy features. A particularly noteworthy result is the consistent performance of $NeuPSL_{individual}$, which either outperforms other models or achieves accuracy within the standard deviation of the best models across most covariance settings (RQ2). Interestingly, as the noise becomes extreme, with covariance values of 50.0 or 100.0, $NeuPSL_{shared}$ surpasses $NeuPSL_{individual}$ and GNN in predictive performance. This observation underscores the potential advantages of parameter sharing in scenarios where the features provide little to no informative signal, highlighting a specific use case where shared parameters can act as a stabilizing factor in noisy environments.

Key Takeaways:

1. There is a trade-off between shared and individual parameters over structure: with lower noise, individual parameters can outperform due to their specificity, while higher noise benefits from the smoothing effect provided by shared parameters. This highlights the inherent advantage of methods like NeuPSL, which can flexibly represent both paradigms.
2. Methods leveraging architectural choices with guaranteed structural reasoning (NeuPSL), rather than relying solely on subsymbolic representations (GNN), demonstrate superior performance when the constraints accurately represent the underlying problem.

7.4 NeSy Pitfalls and Mitigation Strategies

In this final collection of experiments, let’s turn our focus to common pitfalls in neural-symbolic systems and the strategies used to mitigate them. Unlike the previous sec-

tions, which primarily compared NeSy approaches against one another or evaluated their performance relative to purely symbolic or subsymbolic baselines, this section investigates challenges that arise specifically during NeSy inference and learning. The aim here is introspective: to examine NeSy methods in isolation and assess how mitigation strategies can address their inherent shortcomings. Rather than attempting to showcase every potential pitfall, the experiments in this section highlight a representative set of challenges commonly encountered in NeuPSL, offering practical insights into their resolution. The hope is that this collection provides a useful framework for understanding how mitigation strategies can be applied across a range of issues (RQ3). The pitfalls discussed in this section include reasoning shortcuts (Section 5.2.1), contextual label ambiguity (Section 5.3.1), energy loss degenerate solutions (Section 5.3.2), and soft logic challenges (Section 5.3.3).

7.4.1 Reasoning Shortcuts

Reasoning shortcuts (Section 5.2.1) represent a pervasive and often subtle challenge within NeSy methods, arising when a model finds unintended ways to solve a task without fully capturing or learning the underlying concepts. This experiment highlights the issue through a straightforward scenario, providing insights into how reasoning shortcuts emerge and how they can be effectively mitigated. This experiment is over a variation to the Visual Sudoku problem introduced by Augustine et al. (2022) called *Visual Sudoku Puzzle Classification*. In this task, 4×4 Sudoku puzzles are constructed using unlabeled MNIST images, and the model must determine whether a given puzzle is valid—specifically, whether it contains any duplicate digits in any row, column, or square. While the task may seem simple, it introduces significant risks of reasoning shortcuts, as identifying that there are no duplicate digits in any row, column, or square does not mean the model will accurately classify the underlying digits themselves. For this experiment, five random splits of 16/25/50 (train/test/valid) 4×4 Sudoku puzzles were generated. Half of the puzzles in each set are valid, i.e., adhering to Sudoku rules, while the other half includes at least one violation.

Figure 7.6 illustrates the two symbolic models employed to study reasoning shortcuts in the visual sudoku puzzle classification task. Both models follow a DSVar modeling

Figure 7.6: Visual Sudoku Puzzle Classification Reasoning Shortcut Models.

<p>VISUAL SUDOKU PUZZLE CLASSIFICATION MODEL Without MITIGATION (NEUPSL_{shortcut})</p> <p><i># Row, Column, and Box Constraints</i></p> <p>1.0 : IMAGEDIGIT(Puzzle, +X, Y, Number) = 1</p> <p>1.0 : IMAGEDIGIT(Puzzle, X, +Y, Number) = 1</p> <p>1.0 : IMAGEDIGIT(Puzzle, X₁, Y₁, Number) ∧ SAMEBOX(X₁, Y₁, X₂, Y₂) → !IMAGEDIGIT(Puzzle, X₂, Y₂, , Number)</p>
<p>VISUAL SUDOKU PUZZLE CLASSIFICATION MODEL With STRUCTURAL MITIGATION (NEUPSL_{mitigation})</p> <p><i># Row, Column, and Box Constraints</i></p> <p>1.0 : IMAGEDIGIT(Puzzle, +X, Y, Number) = 1</p> <p>1.0 : IMAGEDIGIT(Puzzle, X, +Y, Number) = 1</p> <p>1.0 : IMAGEDIGIT(Puzzle, X₁, Y₁, Number) ∧ SAMEBOX(X₁, Y₁, X₂, Y₂) → !IMAGEDIGIT(Puzzle, X₂, Y₂, , Number)</p> <p><i># Structural Mitigation - Pin First Row of First Puzzle</i></p> <p>1000.0 : FIRSTPUZZLE(Puzzle, X, "0") = IMAGEDIGIT(Puzzle, X, "0")</p>

paradigm (Section 3.2.1) and utilize a constraint loss (Section 4.4.3) distant supervision (Section 4.4.1) learning approach. The first model, NeuPSL_{shortcut}, includes only the standard Sudoku constraints, ensuring no duplicate digits in any row, column, or box. However, this model suffers from a critical issue: there is no enforced correlation between the neural model’s predicted concepts and the true underlying concepts, i.e., digit labels. As a result, while the model may classify puzzles correctly based on surface-level patterns, it fails to learn accurate representations of individual digits. To address this issue, the NeuPSL_{mitigation} model introduces a structural mitigation strategy designed to align predicted concepts with ground truth labels. Specifically, this mitigation involves adding a “pin first-row” rule, which enforces alignment between the predicted digit labels and the actual labels in the first row of the first puzzle. This additional rule provides a structural anchor that helps the model associate the neural predictions with the true concepts. Both models share the same underlying neural architecture.

Table 7.14 presents the average prediction performance for the 4 × 4 visual sudoku puzzle classification task. The results reveal a clear reasoning shortcut in the NeuPSL_{shortcut} model. While its objective value is effectively zero—indicating that the Sudoku constraints are satisfied—its digit and puzzle accuracies are significantly lower than those achieved by NeuPSL_{mitigation}. This highlights that NeuPSL_{shortcut} satisfies the constraints without learning the correct digit representations. In contrast, NeuPSL_{mitigation} achieves substan-

Table 7.14: Test set final objective and puzzle and digit accuracy for NeuPSL models on visual sudoku puzzle classification with and without a reasoning shortcut mitigation.

Method	Accuracy (%)		Final Objective
	Puzzle	Digit	
NeuPSL _{shortcut}	50.00 ± 0.00	23.80 ± 5.74	$2.76 \cdot 10^{-4} \pm 1.88 \cdot 10^{-4}$
NeuPSL _{mitigation}	69.20 ± 2.77	94.90 ± 0.55	42.63 ± 3.43

tially better performance, with a digit classification accuracy of approximately 95%. This improvement comes at the cost of anchoring concepts to known predictions. While structural mitigation is used here, alternative strategies, such as data mitigation (e.g., pretraining the neural model with a few labeled examples), could achieve similar results.

Although this experiment is a simplified example designed to illustrate the occurrence and resolution of reasoning shortcuts, such scenarios are often more complex in real-world applications. For instance, in the ROAD-R dataset, introducing a collective rule such as:

$$1.0 : \text{OBJECT}(\text{Frame}, \text{Box}_1, \text{"Traffic Light"}) \wedge \text{OBJECT}(\text{Frame}, \text{Box}_1, \text{"Red"}) \wedge \\ \text{OBJECT}(\text{Frame}, \text{Box}_2, \text{"Traffic Light"}) \rightarrow \text{OBJECT}(\text{Frame}, \text{Box}_2, \text{"Red"})$$

can inadvertently create a reasoning shortcut. This rule aims to incorporate the concept “if two objects are traffic lights, then they must both be red”. While this can achieve the desired result, the model could satisfy this rule by avoiding predicting the “Traffic Light” class entirely. Identifying such shortcuts in complex datasets can be challenging, but recent work [83] has focused on methods for detecting and addressing these issues.

7.4.2 Contextual Label Ambiguity

Contextual label ambiguity (Section 5.3.1) is one of the most common pitfalls encountered when beginning in NeSy. This issue arises when a NeSy model trains a decomposed structure (Section 4.2.3) neural-symbolic model, but the gradient or label passed from the symbolic component to the neural model lacks sufficient context to be interpretable. For example, in the visual Sudoku problem, a neural model typically processes

only the pixels of an individual square. If a symbolic rule determines that a “blank” square should take on a specific value based on the full Sudoku board, the gradient passed back to the neural model will lack the contextual information necessary to resolve this ambiguity. To explore the implications of this pitfall and potential mitigation strategies, this section examines a modified version of the visual Sudoku problem. For this experiment, 9×9 partially solved Sudoku puzzles were constructed using unlabeled MNIST images, with the objective being to solve the entire board. To address the reasoning shortcut discussed in the previous subsection, a few-shot variant was introduced. In this setting, five labeled examples from each of the nine possible classes were made available for training. The remaining training images remained unlabeled, requiring the model to rely heavily on the Sudoku rules for learning. Five random splits of 20/100/100 (train/validation/test) Sudoku puzzles were generated for evaluation.

This experiment employs the same underlying NeSy NeuPSL model but varies the training data to study its effects. The NeuPSL model is a decomposed DSVar NeSy model, where the neural component predicts digit labels and the symbolic component solves the sudoku rules. To distinguish these models, define them as *NeuPSL_{ambiguous}* and *NeuPSL_{mitigation}*. The *NeuPSL_{ambiguous}* model is trained on data that introduces contextual label ambiguity, while *NeuPSL_{mitigation}* incorporates a mitigation strategy to address this pitfall. To illustrate the differences between these models—stemming primarily from their respective training sets—consider the following 4×4 partially solved Sudoku puzzles constructed using the classes $\{0, 3, 5, 8\}$ and corresponding training datasets.

0	5		3
8	3		0
3			
		3	

$$\begin{aligned}
NeuPSL_{ambiguous} &= \left\{ \begin{array}{cccccccc} \boxed{0}, & \boxed{5}, & \boxed{}, & \boxed{3}, & \boxed{8}, & \boxed{3}, & \boxed{}, & \boxed{0}, \\ \boxed{3}, & \boxed{}, & \boxed{}, & \boxed{}, & \boxed{}, & \boxed{}, & \boxed{3}, & \boxed{} \end{array} \right\} \\
NeuPSL_{mitigation} &= \left\{ \begin{array}{cccccccc} \boxed{0}, & \boxed{5}, & \boxed{3}, & \boxed{8}, & \boxed{3}, & \boxed{0}, & \boxed{3}, & \boxed{3} \end{array} \right\}
\end{aligned}$$

In the above example, the $NeuPSL_{ambiguous}$ receives inconsistent gradients for the same neural input, i.e., each blank square $\boxed{}$ is assigned a different gradient. This ambiguity arises because the symbolic gradient leverages the full structure of the problem, while the neural model’s local input lacks the necessary context to distinguish between squares. Consequently, the neural model struggles to converge on a consistent representation, effectively hindering training as it incorporates noise from these conflicting updates.

Table 7.15: Test set digit accuracy predicted by the neural and symbolic components for NeuPSL models on visual sudoku with and without ambiguous local context.

Method	Prediction Accuracy (%)	
	Neural Digit	Symbolic Digit
$NeuPSL_{ambiguous}$	41.06 ± 5.83	46.66 ± 8.36
$NeuPSL_{mitigation}$	97.19 ± 0.37	98.07 ± 0.41

As shown in Table 7.15, the average prediction performance for the 9×9 visual sudoku task highlights the challenges posed by contextual label ambiguity. Both neural digit accuracy and symbolic digit accuracy are presented after solving the Sudoku board. The $NeuPSL_{ambiguous}$ model suffers from a significant drop in neural digit accuracy, approximately 51 percentage points, compared to $NeuPSL_{mitigation}$. This decline arises because the $NeuPSL_{ambiguous}$ model passes gradients to the blank squares, introducing ambiguity and making it difficult for the neural model to distinguish between classes. As a result, the neural model struggles to learn meaningful representations, ultimately hurting its overall performance.

While this is a simplified example to illustrate the pitfall and its mitigation strategy, such issues in real-world applications can be more complex and harder to identify. For instance, in dialog structure induction tasks, incorporating external knowledge about the sequential order of utterances in a conversation can lead to similar contextual label ambiguity. This occurs if the neural model does not have access to the full dialog as input.

7.4.3 Energy Loss Degenerate Solutions:

This experiment investigates a common degenerate solution of energy learning (Section 5.3.2), which is the most typical value-based learning loss. In particular, the one being studied here arises when the symbolic parameters create an energy configuration that cannot differentiate between target assignments, leading to what is known as a *collapsed energy function*. A common instance of this issue in NeuPSL occurs when all symbolic parameters are set to zero, i.e., $w_{sy} = \mathbf{0}$. This scenario results in a collapsed energy function because the symbolic potentials, which rely on their interaction with the symbolic parameters, fail to contribute meaningful structure or gradients to the learning process. Consequently, the learning mechanism cannot effectively optimize or encode the problem’s constraints. To illustrate this phenomenon, the experiment evaluates its impact using citation network datasets. Two NeuPSL architectures are compared: NeuPSL_{degenerate}, which lacks a simplex constraint on the symbolic weights, and NeuPSL_{mitigation}, which incorporates a simplex constraint.

Table 7.16: Test set accuracy for the NeuPSL models with and without the zeroed weights degenerate solution.

Method	Citeseer (Accuracy)	Cora (Accuracy)
NeuPSL _{degenerate}	16.74 ± 0.70	14.24 ± 1.46
NeuPSL _{mitigation}	68.48 ± 1.22	81.22 ± 0.79

Figure 7.7 provides the final symbolic parameters learned by NeuPSL_{degenerate} and NeuPSL_{mitigation}. The comparison clearly demonstrates that without the weight sim-

Figure 7.7: Citeseer and Cora learned models with and without parameter simplex constraints.

<p>CITSEER LEARNED PARAMETERS WITH Weight SIMPLEX (<i>NeuPSL_{mitigation}</i>)</p> <p>1.00 : $\text{NEURAL}(\text{Paper}, \text{Label}) = \text{CATEGORY}(\text{Paper}, \text{Label})$</p> <p>0.44 : $\text{LINK}(\text{Paper1}, \text{Paper2}) \wedge \text{CATEGORY}(\text{Paper1}, \text{Label}) \rightarrow \text{CATEGORY}(\text{Paper2}, \text{Label})$</p> <p>$\text{CATEGORY}(\text{Paper}, +\text{Label}) = 1.$</p>
<p>CITSEER LEARNED PARAMETERS Without WEIGHT SIMPLEX (<i>NeuPSL_{degenerate}</i>)</p> <p>0.00 : $\text{NEURAL}(\text{Paper}, \text{Label}) = \text{CATEGORY}(\text{Paper}, \text{Label})$</p> <p>0.00 : $\text{LINK}(\text{Paper1}, \text{Paper2}) \wedge \text{CATEGORY}(\text{Paper1}, \text{Label}) \rightarrow \text{CATEGORY}(\text{Paper2}, \text{Label})$</p> <p>$\text{CATEGORY}(\text{Paper}, +\text{Label}) = 1.$</p>
<p>CORA LEARNED PARAMETERS With WEIGHT SIMPLEX (<i>NeuPSL_{mitigation}</i>)</p> <p>1.00 : $\text{NEURAL}(\text{Paper}, \text{Label}) = \text{CATEGORY}(\text{Paper}, \text{Label})$</p> <p>0.46 : $\text{LINK}(\text{Paper1}, \text{Paper2}) \wedge \text{CATEGORY}(\text{Paper1}, \text{Label}) \rightarrow \text{CATEGORY}(\text{Paper2}, \text{Label})$</p> <p>$\text{CATEGORY}(\text{Paper}, +\text{Label}) = 1.$</p>
<p>CORA LEARNED PARAMETERS Without WEIGHT SIMPLEX (<i>NeuPSL_{degenerate}</i>)</p> <p>0.00 : $\text{NEURAL}(\text{Paper}, \text{Label}) = \text{CATEGORY}(\text{Paper}, \text{Label})$</p> <p>0.00 : $\text{LINK}(\text{Paper1}, \text{Paper2}) \wedge \text{CATEGORY}(\text{Paper1}, \text{Label}) \rightarrow \text{CATEGORY}(\text{Paper2}, \text{Label})$</p> <p>$\text{CATEGORY}(\text{Paper}, +\text{Label}) = 1.$</p>

plex constraint, $\text{NeuPSL}_{\text{degenerate}}$ produces an uninformative model where all MAP states are indistinguishable. Table 7.16 summarizes the predictive performance of the models depicted in Figure 7.7. The $\text{NeuPSL}_{\text{degenerate}}$ model, having zero weights and thus failing to distinguish between MAP states, results in inference that is trivial. The final predictions correspond directly to the initial variable assignments, which, in this case, were random. As a result, the $\text{NeuPSL}_{\text{degenerate}}$ model exhibits performance that is effectively random, highlighting the critical role of the weight simplex constraint in preventing this degenerate solution and enabling the model to make meaningful predictions.

7.4.4 Soft Logic Pitfalls in NeuPSL

This final pitfall experiment delves into common challenges in soft logic-based NeSy inference and learning (Section 5.3.3), with a specific focus on NeuPSL-related issues. These challenges stem from the soft logic relaxation employed in NeuPSL, Łukasiewicz logic. This relaxation introduces a “satisfaction hinge,” where rules can be satisfied at any value between 0.7 and 1 rather than strictly at the maximum value of 1. This hinge creates

flat gradient regions. The issue is further compounded in decomposed tasks, where the neural component relies exclusively on symbolic structure for learning. Without sufficient labeled data, gradients often converge around the average, leading to poor generalization and suboptimal neural representations. To illustrate this pitfall, the experiment focuses on the MNIST-Add k problem using the same decomposed DSVar NeuPSL_{default} model introduced earlier for this dataset. To address or mitigate this issue, a second model, NeuPSL_{mitigation}, is introduced, where the digit classifier is pretrained using common self-supervision techniques from neural network literature. Specifically, the digit classifier backbone is pretrained using the SimCLR self-supervised learning framework [24]. Positive pairs for contrastive pretraining are generated through augmentations such as cropping, rotation, and color jittering. Both models are evaluated on five random splits.

Table 7.17: Test set accuracy for the NeuPSL models with and without a pretrained backbone.

Method	MNIST-Add1 (Accuracy)	MNIST-Add2 (Accuracy)
NeuPSL _{default}	82.58 \pm 02.56	56.94 \pm 06.33
NeuPSL _{mitigation}	93.80 \pm 1.12	87.92 \pm 1.63

Table 7.17 presents the results for the MNIST-Add k experiments, comparing NeuPSL_{default} and NeuPSL_{mitigation}. While NeuPSL_{default} demonstrates reasonable performance, the addition of the pretraining step in NeuPSL_{mitigation} leads to a significant improvement. This enhanced performance surpasses the results achieved by the binarized approach of DeepProbLog (Table 7.10). It is worth noting that this mitigation strategy—employing a pretraining step—is not exclusive to NeuPSL. Similar strategies could likely yield performance improvements for other NeSy systems, including DeepProbLog. The key insight here is that the pretraining step effectively mitigates NeuPSL’s averaging and soft logic relaxation pitfalls, enabling it to perform comparably to NeSy systems specifically designed for this type of task.

Chapter 8

Related Work

The integration of symbolic knowledge and reasoning with neural networks has a rich and extensive history, with recent interest particularly over the past decade. This dissertation establishes a foundation for NeSy AI, offering a formal language and conceptual framework for developing, analyzing, and communicating NeSy approaches. In doing so, it discusses or touches upon three critical areas: the diversity of existing NeSy approaches, the taxonomies and organizational principles developed within NeSy AI, and the integration of energy-based models as a unifying framework for modeling neural and symbolic interactions. This chapter reviews the literature within these domains, touching on key aspects retentive to this dissertation.

8.1 Neural-Symbolic Approaches

While this dissertation establishes a foundational framework for NeSy AI, it focuses primarily on systems aligned with the neural as symbolic parameter and neural as symbolic variable architectural axioms. Since this dissertation introduces a novel NeSy approach designed for end-to-end differentiable learning, the discussion will concentrate on systems that similarly emphasize differentiable integration of neural and symbolic components. Accordingly, this section highlights and examines prominent approaches that exemplify these paradigms.

8.1.0.1 Neural as Symbolic Parameter

The neural as symbolic parameter paradigm leverages neural networks to parameterize symbolic components in which the neural model has indirect control over the symbolic programs' prediction. Xu et al. (2018) introduced a loss function derived from probabilistic logic semantics to encode domain knowledge directly into the learning process. Similarly, Yang, Ishay, and Lee (2020) and Manhaeve et al. (2021) proposed NeurASP and DeepProbLog, respectively, which compile tractable probabilistic logic programs into differentiable functions. These frameworks allow symbolic reasoning to guide neural predictions while maintaining compatibility with probabilistic logic semantics. More recently, Maene and Raedt (2024) introduced DeepSoftLog, an extension of ProbLog that incorporates embedded terms, transitioning from fuzzy to probabilistic semantics. Ahmed et al. (2022) proposed Semantic Probabilistic Layers (SPLs), which compile knowledge and logic into differentiable functions using the semantics of probabilistic circuits (PCs) [25]. Cohen, Yang, and Mazaitis (2020) proposes TensorLog, a probabilistic first-order logic framework that compiles tractable probabilistic logic programs into differentiable layers, allowing for efficient learning and inference. As Cohen, Yang, and Mazaitis (2020) observed, querying many probabilistic logic frameworks involves solving the weighted model counting (WMC) problem, which is $\#P$ -complete, or addressing the MAXSAT problem, which is NP-hard [132]. Since deep neural networks operate in polynomial time relative to their size, general logic queries or MAXSAT solving cannot be implemented in polynomial time unless $\#P=P$ or $NP=P$. To address these computational challenges, researchers have developed more efficient differentiable reasoning systems by limiting logic to tractable families [28, 7, 77] or employing approximate inference techniques [132, 80, 128].

8.1.0.2 Neural as Symbolic Variable

The neural as symbolic variable paradigm leverages neural networks to parameterize symbolic components in which the neural model has direct control over the symbolic programs' prediction. Demeester, Rocktäschel, and Riedel (2016) incorporates domain knowledge and common sense into natural language and knowledge base representations by

encouraging partial orderings over embeddings through regularization of the learning loss. Similarly, Rocktäschel and Riedel (2017) leverages knowledge encoded as a differentiable loss derived from logical rules to train a matrix factorization model for relation extraction. Diligenti, Roychowdhury, and Gori (2017) employs fuzzy logic to quantify violations of constraints in the model’s output, which is minimized during the learning process. Wang et al. (2019) integrates logical reasoning with deep learning by introducing a differentiable smoothed approximation to a maximum satisfiability (MAXSAT) solver, enabling logical reasoning within neural models. The Logic Tensor Network (LTN) framework introduced by Badreddine et al. (2022) uses neural network predictions to parameterize functions that represent symbolic relations with real-valued or fuzzy logic semantics. These logic functions are aggregated to define a satisfaction level, and predictions are derived by evaluating the truth value of all possible outputs and selecting the highest-valued configuration. Badreddine, Serafini, and Spranger (2023) extends LTNs by introducing fuzzy operators grounded in logarithmic space, providing more efficient and effective formulations for end-to-end training. Amos and Kolter (2017) integrates linearly constrained quadratic programming (LCQP) problems into neural networks via the OptNet framework, demonstrating that LCQP solutions are differentiable with respect to program parameters. Building on this work, Agrawal et al. (2019) applies domain-specific languages (DSLs) to define LCQP program layers, simplifying the specification of knowledge and constraints for optimization. Vlastelica et al. (2020) proposes a method for computing gradients of mixed-integer linear programs by introducing a continuous interpolation of the program’s objective, supporting integer constraints and approximating gradients for program outputs. In contrast, Cornelio et al. (2023) takes a reinforcement learning approach to mathematical programming, interpreting neural model predictions as states in a Markov decision process. Actions are taken to identify and address constraint violations, and the REINFORCE algorithm [135] is used to train the system end-to-end without requiring backpropagation through the solver. Recent advancements in this area include Giunchiglia et al. (2023), who introduced a dataset for autonomous event detection with embedded logical requirements, demonstrating that such requirements improve generalization. Stoian, Giunchiglia, and Lukasiewicz (2023) further validates this approach by showing enhanced model performance when logical re-

quirements are incorporated into training.

8.2 Taxonomies of NeSy Approaches

The field of neural-symbolic AI encompasses a diverse range of approaches developed over the past two decades to integrate neural networks with symbolic reasoning. In the previous section, I discussed this wide range of approaches, showcasing their generality and versatility from the perspective of the axioms of integration (Section 2). Recently, in the last couple of years, the NeSy community has shifted toward organizing these approaches and developing more comprehensive theoretical frameworks. This effort has led to the emergence of several taxonomies that span various dimensions, including the representation of symbolic knowledge, the interaction between neural and symbolic components, learning and reasoning mechanisms, pitfalls, system libraries, and connections to related fields. These taxonomies are particularly relevant to this dissertation, which aims to establish a foundational framework for NeSy AI. This work provides the groundwork from which these taxonomies can further develop and evolve. In essence, the approach taken here is orthogonal to the existing taxonomies, complementing and supporting them rather than competing with or replacing them. In fact, the majority of the foundational principles introduced in this dissertation are designed to accommodate and reinforce a wide range of these taxonomies, ensuring their continued relevance and applicability within the NeSy field.

1. **Representation of Symbolic Knowledge:** A key dimension of NeSy taxonomies is the representational type and expressiveness of symbolic knowledge within hybrid NeSy approaches. Marra et al. (2024) provides a comprehensive survey that highlights the role of principled symbolic knowledge representations, drawing connections between NeSy AI and the closely related field of statistical relational learning (SRL) [56]. This includes widely used SRL symbolic symantics such as probabilistic logics (ProbLog [39]) and soft logic relaxations such as Łukasiewicz logic (Probabilistic Soft Logic (PSL) [11]). Beyond SRL, other works focus on specific aspects of symbolic logics, particularly their differentiability and the challenges arising in hybrid systems.

For example, Krieken (2024) and van Krieken, Acar, and van Harmelen (2022) investigate fuzzy and probabilistic symbolic logics, analyzing their integration into neural models and the associated trade-offs. Evans and Grefenstette (2018) further explores probabilistic symbolic reasoning.

2. **Neural-Symbolic Integration Patterns:** Another key dimension of NeSy taxonomies focuses on the common integration patterns between neural and symbolic components. These patterns describe how a neural network and symbolic systems are combined at a high level. It is important to distinguish these works from the architectural axioms introduced in Chapter 2. While these taxonomies focus on describing common integration patterns, they do not address the detailed interactions between neural and symbolic components, such as the flow of values or gradients across these components. Bekkum et al. (2021) and Mossakowski (2022) introduce neural-symbolic design patterns that articulate these integration strategies through modular building blocks, including instances, models, processes, and actors. Their work provides a structured framework and visual tools to represent and analyze the architecture of neural-symbolic systems, facilitating a clearer understanding and communication of their design. Additionally, Yu et al. (2023) and Marra et al. (2024) propose taxonomies that investigate neural-symbolic integration from the perspective of inference and learning processes.
3. **Learning and Reasoning Mechanisms:** Several taxonomies delve into the learning and reasoning mechanisms that underpin NeSy systems. d’Avila Garcez et al. (2019) and Dickens et al. (2024) investigate probabilistic and bilevel optimization techniques, proposing unified frameworks that integrate learning and reasoning processes. Krieken (2024) offers a detailed exploration of optimization methods specifically tailored for neuro-symbolic learning systems, with a particular focus on probabilistic logics and their practical applications. Additionally, Marra et al. (2024) examines learning and reasoning by drawing parallels to statistical relational learning (SRL) [56], highlighting shared principles and techniques between these closely related fields.

4. **Pitfalls in NeSy:** Although formal taxonomies for NeSy pitfalls have not yet been established, this dissertation represents an initial effort to organize and analyze the challenges faced within these systems. Among these challenges, reasoning shortcuts—a specific subcategory of pitfalls—have recently garnered attention in the NeSy research community. Marconato et al. (2023) characterizes common reasoning shortcuts in NeSy systems. Building on this, Marconato, Teso, and Passerini (2023) and Marconato et al. (2024) propose strategies for mitigating these shortcuts, including techniques to make model aware of there reasoning shortcuts. Furthermore, Bortolotti et al. (2025) introduces a benchmark suite for evaluating concept quality and reasoning shortcuts, providing a practical framework to study and refine NeSy systems. Together, these efforts mark an important step toward the development of structured pitfall taxonomies in NeSy, laying the groundwork for future research in this area.
5. **NeSy Libraries:** Recent efforts have focused on standardizing mainstream NeSy systems through the development of a universal library. Krieken et al. (2024) introduced ULLER, a unified language specifically designed to represent and formalize major NeSy systems. ULLER aims to provide a standardized framework that enables consistent, modular, and interoperable implementations of NeSy approaches. Its long-term goal is to foster the development of a shared Python library, facilitating collaboration, reproducibility, and the seamless integration of diverse NeSy methodologies into practical applications.
6. **Connections to Related Fields:** Given the broad scope of NeSy AI, it intersects with numerous theoretical fields, either fully or partially encompassing their methodologies and concepts. As a result, there has been a concerted effort to establish connections with these related fields and engage with their respective research communities. Marra et al. (2024) explores the relationship between NeSy systems and statistical relational learning (SRL), drawing parallels between popular SRL approaches and NeSy methodologies. Similarly, Zhang et al. (2021) investigates the connections between NeSy systems and knowledge graphs, while Lamb et al. (2020)

examines their relationship with graph neural networks (GNNs). These studies showcase the potential for NeSy AI to bridge gaps across disciplines and leverage advances in these related fields.

8.3 Energy-Based Models (EBMs)

Throughout this dissertation, I leverage Energy-Based Models (EBMs) [74] as the foundation for the universal NeSy Language introduced in Chapter 3 (NeSy-EBMs). EBMs define a scalar-valued energy function, $E : \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}$, which quantifies the compatibility between observed (input) variables $\mathbf{x} \in \mathcal{X}$ and target (output) variables $\mathbf{y} \in \mathcal{Y}$. States with lower energy values represent higher compatibility. One of the key strengths of EBMs is their generality, as they can model complex dependencies and perform density estimation by defining conditional, joint, and marginal Gibbs distributions over the energy function:

$$P(\mathbf{y}|\mathbf{x}) := \frac{e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}} e^{-\beta E(\hat{\mathbf{y}},\mathbf{x})}}, \quad (8.1)$$

$$P(\mathbf{y}, \mathbf{x}) := \frac{e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}, \hat{\mathbf{x}} \in \mathcal{X}} e^{-\beta E(\hat{\mathbf{y}},\hat{\mathbf{x}})}}, \quad (8.2)$$

$$P(\mathbf{x}) := \frac{\int_{\hat{\mathbf{y}} \in \mathcal{Y}} e^{-\beta E(\mathbf{y},\mathbf{x})}}{\int_{\hat{\mathbf{y}} \in \mathcal{Y}, \hat{\mathbf{x}} \in \mathcal{X}} e^{-\beta E(\hat{\mathbf{y}},\hat{\mathbf{x}})}}. \quad (8.3)$$

The universality of Gibbs distributions is a primary motivation for adopting the EBM framework. Any density function can, in principle, be expressed using an appropriate energy function E . This universality positions EBMs as a unifying framework, bridging probabilistic and non-probabilistic approaches, and supporting a wide range of tasks in both generative and discriminative modeling.

Energy-Based Models (EBMs) have been widely applied across machine learning to model complex data distributions and make predictions. Some of the earliest and most influential examples include the Boltzmann machine [2, 110] and the Helmholtz machine [38], both of which laid the groundwork for the development of modern EBMs. In a seminal contribution, Hinton (2002) demonstrated the utility of EBMs in constructing mixture-of-expert models. This approach combines multiple simpler distributions to approximate

a single, complex distribution by multiplying the individual distributions and applying renormalization. These early works highlighted the flexibility and power of EBMs for representing and learning complex dependencies in high-dimensional data, establishing their relevance for a wide range of machine learning applications.

In recent years, EBMs have proven effective in discriminative tasks [60, 76]. Grathwohl et al. (2020) introduced the Joint Energy-Based Model (JEM), which reinterprets discriminative classifiers as EBMs and uses a likelihood-based loss to train on both labeled and unlabeled data. This approach improves accuracy, robustness, calibration, and out-of-distribution detection while unifying generative and discriminative learning under the EBM framework. Expanding on this, Liu et al. (2020) developed an EBM specifically for out-of-distribution detection. Their approach employs a purely discriminative training objective and uses unnormalized energy scores to effectively identify out-of-distribution examples. This method achieves state-of-the-art performance and further highlights the practical advantages of EBMs in tasks requiring robustness and uncertainty estimation.

Beyond discriminative modeling, the EBM framework has become a powerful tool in generative modeling, offering an alternative to traditional approaches [144, 50, 51]. Zhao, Mathieu, and LeCun (2017) introduced Energy-Based Generative Adversarial Networks (EBGANs), which reinterpret the GAN discriminator as an energy function assigning low energy to points near the data manifold. This approach addresses instability issues in GAN training and enables more robust learning across diverse architectures and loss functions. Building on this, Du and Mordatch (2019) demonstrated the direct use of EBMs for generative modeling, highlighting their simplicity, stability, and flexibility. They showed that EBMs achieve performance comparable to modern GANs while excelling in tasks such as out-of-distribution detection and adversarially robust classification. Their work also emphasized the compositionality of EBMs, enabling modular and interpretable generative processes. More recently, Du et al. (2023) extended EBMs to diffusion models, proposing an energy-based parameterization that enhances flexibility and expressiveness in compositional generation.

These examples underscore the generality and broad applicability of NeSy-EBMs across diverse tasks and domains. While the primary focus of this dissertation is not on

the specific methods for training and learning these models, it is important to note that a wide range of probabilistic and non-probabilistic approaches have been developed for this purpose [63, 134, 50, 60]. For a comprehensive review of related work on training methodologies for these systems, I direct the reader to the detailed discussion provided by Dickens (2024).

Chapter 9

Future Work and Limitations

In this chapter, I reflect on the work presented in this dissertation, acknowledging the inherent limitations of my contributions to the foundational aspects of NeSy AI while highlighting a collection of promising directions for future research. This discussion is organized around the key foundational elements introduced: the *axioms of integration* (Part I), the *universal language* (Part II), the *design principles* (Part III), and the *general implementations* (Part IV). The goal of this chapter is to provide a clear roadmap for steps to address these limitations and a guide to expand upon the foundational work presented here, paving the way for further advancements in the field of NeSy AI.

9.1 Neural-Symbolic Axioms of Integration:

As emphasized in the introduction, while neural-symbolic research consistently employs foundational integration techniques, the field still lacks a cohesive and explicitly defined set of axioms to unify and guide these efforts. In response, in Chapter 2, I proposed four key architectural axioms that serve as the *current* cornerstones for NeSy research. While these axioms provide a unified perspective on integration, I acknowledge that:

1. **Alternative Perspectives:** The architectural axioms represent a single perspective on neural-symbolic integration, and alternative formulations may offer clearer or more concise frameworks.

2. Incomplete Coverage: Even within the architectural axioms presented, not all forms of neural-symbolic approaches are currently represented.

Regarding the first point, a set of alternative viewpoints could prove valuable for further developing the theory. For instance, defining and describing the axioms of integration from the perspective of systems could offer new insights and broaden scope of the community. Moreover, alternative organizations to the current set of architectural axioms could provide a more succinct categorization. For instance, the categories *neural as symbolic variables* (Section 2.3.4) and *neural as symbolic parameters* (Section 2.3.3) could be merged into a broader and more inclusive category, such as *neural as differentiable symbolic systems*. Regardless of the specific form these axioms take, I argue that developing, formalizing, and teaching such axioms represents a critical next step for advancing the NeSy field. Moreover, the foundational principles and organizational structure of NeSy research must be presented in a concise, accessible, and generalizable manner to support future researchers, including those from outside the immediate NeSy community. This will ensure the continued growth and adoption of NeSy methodologies across diverse domains.

Regarding the second point, the proposed architectures were deliberately aligned with the universal language presented in Chapter 3. This alignment was intentional, designed to create a cohesive narrative from architectural foundations to universal language, principles, and general implementation. However, it is important to recognize that additional integration strategies exist within the NeSy field that may not easily fit within the presented axioms. For instance, three potential axioms not explicitly addressed in this thesis are *extracting symbolic knowledge from neural models*, *neural as symbolic structure*, and *symbolic as neural input*. The first potential additional axiom could represent a well-established research direction aimed at extracting symbolic representations from the black-box nature of neural networks. This process enhances interpretability, transparency, and knowledge transfer, serving as a critical bridge between neural and symbolic systems. The second axiom, *neural as symbolic structure*, introduces a missing dimension to the framework by capturing scenarios where neural models define symbolic structures in differentiable spaces. This approach complements the existing trio of differentiable neural-symbolic categories (*neural as differentiable variables* Section 2.3.4 and *neural as*

differentiable parameters Section 2.3.3). The final axiom, *symbolic as neural input*, formalizes the reverse direction of integration, where the NeSy method involves the composition of symbolic outputs being used as inputs to the neural component.

9.2 Universal Neural-Symbolic Language

The universal neural-symbolic language presented in this dissertation is closely tied to the axioms of integration, meaning it should be flexible enough to support any collection of integration axioms. While NeSy-EBMs and the associated modeling paradigms introduced in Chapter 2 encompass many of the architectural axioms, it is important to note that if the axioms are expanded or modified, the universal language must evolve to accommodate these changes. As such, I acknowledge the following limitations and areas for future research:

1. **Recursive NeSy-EBMs:** The NeSy-EBM framework introduced was designed as a direct composition of neural and symbolic components. However, systems that incorporate iterative or recursive interactions between neural and symbolic components would require an expanded definition.
2. **Missing Modeling Paradigms:** As presented in Section 5.1, the modeling paradigms presented in Section 3.2 are not exhaustive.
3. **Incomplete Translation of NeSy Approaches:** This dissertation only translates three prominent NeSy approaches into the NeSy-EBM formulations (Section 3.3). A more concerted effort is needed to categorize and formalize the current plethora of NeSy methods within this framework.

Regarding the first point, expanding the theoretical framework of NeSy-EBMs to support recursive interactions—where neural outputs feed into symbolic systems, which in turn influence subsequent neural components, and so on—is a highly promising direction for future work. Such iterative integration holds the potential to model more complex, dynamic systems that require deeper interaction between neural and symbolic components. Recent

efforts by [15] have begun to categorize patterns of neural-symbolic integration, primarily from the perspective of inference and learning. While valuable, these efforts highlight the need for a more formal and comprehensive framework to accommodate recursive interactions. Achieving this will require significant theoretical advancements, including extending existing proofs in inference and learning theory to address the complexities introduced by iterative integration.

Regarding the second point, as the set of architectural axioms and theoretical foundations of NeSy expands, the corresponding set of modeling paradigms must also grow and be rigorously studied. For example, Section 5.1 introduced two alternative NeSy-EBM modeling paradigms: *unfixed deep symbolic variables* (Section 5.1.1) and *deep symbolic operations* (Section 5.1.2). Furthermore, exploring combinations of existing NeSy-EBM modeling paradigms will be critical for building fully integrated neural-symbolic AI systems. For instance, integrating *deep symbolic variables* (Section 3.2.1) with *deep symbolic parameters* (Section 3.2.2) could enable the development of more sophisticated and multi-modal NeSy learning systems.

Regarding the third point, the systems formulated in this dissertation were selected as they represent mainstream approaches within the NeSy community. While these serve as valuable examples, there remains a need to extend this effort to incorporate a broader range of NeSy approaches. Expanding the framework to include a more comprehensive set of methods will not only help organize the field but also clarify distinctions between systems that share similar theoretical foundations. Such efforts would promote greater understanding, reduce ambiguity, and foster further advancements in the development and application of NeSy systems.

9.3 Neural-Symbolic Design Principles:

Similarly to the previous section, the expansion of the axioms of integration and the universal neural-symbolic language will inevitably lead to new dimensions for the set of design principles. However, unlike the previous sections, I argue that the primary limitation of the neural-symbolic design principles lies in their current incompleteness.

In this dissertation, I have presented a selection of the most fundamental principles and pitfalls encountered throughout my research and reading. While these principles provide a solid foundation, they are by no means exhaustive. For future research, these design principles should be expanded to study:

1. **Universal Design Principles:** Developing a more comprehensive set of general design principles applicable across a wide range of NeSy approaches that can be used for a general researcher within and outside of NeSy.
2. **Domain-Specific Design Principles:** Creating collections of design principles tailored to specific subsets of NeSy approaches, such as fuzzy logic pitfalls or probabilistic logic inference techniques.

In general, developing a comprehensive taxonomy of universal and domain-specific design principles will not only accelerate research progress but also lower the barrier to entry for newcomers to the NeSy field. Furthermore, it will provide a valuable resource for experienced NeSy researchers, enabling them to identify connections and commonalities across different NeSy theories. This, in turn, will foster greater collaboration and drive innovation within the broader NeSy research community.

9.4 A General Neural-Symbolic Implementation

In this work, I introduced a novel and practical NeSy implementation, *Neural Probabilistic Soft Logic* (NeuPSL), and provided a comprehensive set of experimental evaluations. While this method supports a broad range of architectural axioms, NeSy-EBM modeling paradigms, and design principles, and has been applied to a wide range of tasks, I acknowledge the following limitations:

1. **Inability to Represent Every NeSy-EBM:** NeuPSL cannot represent every NeSy-EBM due to its design focus and specific functionalities.
2. **Need for Additional Real-World Experiments:** NeuPSL has demonstrated its capabilities across various tasks, but further real-world experiments are necessary to

validate practical impact in diverse domains.

In regards to the first point, as reiterated in Section 6.3, NeuPSL defines a set of core functionalities that should be supported by NeSy implementations while maintaining flexibility for future extensions. However, this design inherently limits its ability to represent all NeSy-EBMs. NeuPSL currently defines weighted sums of potentials based on arithmetic, logic, and Łukasiewicz real-logic semantics, which provide a solid foundation but are not exhaustive. Future work could explore the integration of alternative soft logic formulations to expand the range of representable NeSy-EBMs. Additionally, NeuPSL is primarily designed for non-probabilistic inference tasks, such as prediction, ranking, and detection, due to the computational challenges of marginal distributions and the Gibbs partition function. Expanding NeuPSL to support marginal inference instead of maximum a posteriori (MAP) inference represents an exciting direction for future research, potentially enabling applications in more probabilistic and uncertainty-aware domains.

While NeuPSL has been tested in controlled experimental settings, further validation through real-world applications is essential to fully assess its scalability, robustness, and practical utility. Deploying NeuPSL in domains such as healthcare, autonomous systems, or natural language processing could provide valuable insights into its strengths and limitations in complex, real-world scenarios. Furthermore, a more detailed exploration of the trade-offs between the major NeSy systems to better understand when and where to use these approaches is another direction for future work.

Chapter 10

Conclusion

As the world shifts from merely exploring the potential of AI to increasingly relying on its applications, the urgency for responsible deployment and careful interpretation of these models outputs has become profoundly evident. Neural-symbolic AI stands as a field uniquely equipped to tackle some of the most pressing challenges, including the promise of interpretability, the enforcement of constraints, and the assurance of consistent, reliable predictions. Yet, despite its immense promise, NeSy AI remains a relatively nascent and fragmented field, often defined by ad-hoc implementations that lack cohesion and standardization. As its development continues to grow, so too does the need for a principled foundation.

In this dissertation, I propose and contribute to a formalized foundation for neural-symbolic AI research through four key milestones: *axioms of integration*, *universal neural-symbolic language*, *design principles*, and *general and principled implementations*. Together, these milestones establish a cohesive roadmap for unifying the theoretical and practical underpinnings of NeSy AI. To address the fragmentation within the field, I take a first step in defining four foundational architectural axioms that serve as cornerstones for current NeSy research: *symbolic as neural structure*, *sampling neural for symbolic*, *neural as symbolic variable*, and *neural as symbolic parameter*. Recognizing the community’s emphasis on logic-based NeSy approaches, I provide detailed examples for each axiom, grounded in symbolic logic, to offer familiar and tangible scenarios that elucidate these principles. To

formalize these integration strategies, I introduce Neural-Symbolic Energy-Based Models (NeSy-EBMs) as a unifying mathematical framework. Within this framework, I develop three essential modeling paradigms—*deep symbolic variables*, *deep symbolic parameters*, and *deep symbolic potentials*—that enable the practical implementation of the architectural axioms. Building on this foundation, I categorize and define a comprehensive set of NeSy design principles, encompassing inference and learning techniques, strategies, and pitfalls. Finally, I introduce Neural Probabilistic Soft Logic (NeuPSL), a novel and scalable framework that implements a collection of the design principles, modeling paradigms, and architectural axioms. NeuPSL demonstrates its versatility and effectiveness across diverse real-world applications, including event detection, dialog structure induction, and logic-based question answering. Additionally, I provide a detailed comparative analysis of NeuPSL alongside four prominent NeSy approaches, emphasizing their respective strengths and limitations. This comparison not only highlights the unique benefits of NeuPSL but also illuminates broader insights into the practical challenges and opportunities within NeSy systems. Finally, I illustrate how NeuPSL addresses a collection of common NeSy pitfalls, offering actionable strategies for their mitigation.

Looking to the future, the integration of symbolic reasoning will be indispensable as AI increasingly influences every aspect of daily life. Neural models alone, despite their remarkable adaptability and learning capabilities, are insufficient for addressing the transparency, robustness, and ethical safeguards required in sensitive and high-stakes domains such as healthcare, autonomous systems, and legal decision-making. Reflecting on Patrick Winston’s words from the start of this dissertation: “Instead of looking for a ‘right way,’ the time has come to build systems out of diverse components, some connectionist and some symbolic.” While perhaps ahead of its time, this vision has grown into an urgent imperative. I contend that the future of AI lies in hybrid approaches—combining the nuanced pattern recognition and learning power of neural models with the rigor, interpretability, and logical consistency of symbolic reasoning. By merging these paradigms, we can create AI systems that are not only powerful but also transparent, explainable, and safe, setting a new standard for trustworthy and ethical artificial intelligence for years to come.

Bibliography

- [1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. <https://www.tensorflow.org/>. 2015.
- [2] David Ackley, Geoffrey Hinton, and Terrence Sejnowski. “A Learning Algorithm for Boltzmann Machines”. In: *Cognitive Science* 9.1 (1985), pp. 147–169.
- [3] AshKay Agrawal et al. “Differentiable Convex Optimization Layers”. In: *NeurIPS*. 2019.
- [4] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. “Semantic Strengthening of Neuro-Symbolic Learning”. In: *AISTATS*. 2023.
- [5] Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. “A Pseudo-Semantic Loss for Autoregressive Models with Logical Constraints”. In: *NeurIPS*. 2023.
- [6] Kareem Ahmed et al. “Neuro-Symbolic Entropy Regularization”. In: *UAI*. 2022.
- [7] Kareem Ahmed et al. “Semantic Probabilistic Layers for Neuro-Symbolic Learning”. In: *NeurIPS*. 2022.
- [8] Brandom Amos and J. Zico Kolter. “OptNet: Differentiable Optimization as a Layer in Neural Networks”. In: *ICML*. 2017.
- [9] Peter Anderson et al. “On Evaluation of Embodied Navigation Agents”. In: *CoRR* abs/1807.06757 (2018).
- [10] Eriq Augustine et al. “Visual Sudoku Puzzle Classification: A Suite of Collective Neuro-Symbolic Tasks”. In: *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*. 2022.

- [11] Stephen Bach et al. “Hinge-Loss Markov Random Fields and Probabilistic Soft Logic”. In: *Journal of Machine Learning Research (JMLR)* 18.1 (2017), pp. 1–67.
- [12] Sebastian Bader and Pascal Hitzler. “Dimensions of Neural-symbolic Integration - A Structured Survey”. In: *arXiv* (2005).
- [13] Samy Badreddine, Luciano Serafini, and Michael Spranger. “logLTN: Differentiable Fuzzy Logic in the Logarithm Space”. In: *arXiv* (2023).
- [14] Samy Badreddine et al. “Logic Tensor Networks”. In: *AI* 303.4 (2022), p. 103649.
- [15] Michael van Bekkum et al. “Modular Design Patterns for Hybrid Learning and Reasoning Systems: A Taxonomy, Patterns and Use Cases”. In: *Applied Intelligence* 51.9 (2021), pp. 6528–6546.
- [16] Tarek R. Besold et al. “Neural-Symbolic Learning and Reasoning: A Survey and Interpretation”. In: *Neuro-Symbolic Artificial Intelligence: The State of the Art* (2022).
- [17] Samuele Bortolotti et al. “A Neuro-Symbolic Benchmark Suite for Concept Quality and Reasoning Shortcuts”. In: *NeurIPS*. 2025.
- [18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [19] James Bradbury et al. *JAX: Autograd and XLA*. <https://github.com/google/jax>. 2018.
- [20] Giovanni Campagna et al. “Zero-Shot Transfer Learning with Synthesized Data for Multi-Domain Dialogue State Tracking”. In: *ACL*. 2020.
- [21] Nicolas Carion et al. “End-to-end Object Detection with Transformers”. In: *ECCV*. 2020.
- [22] Angel Chang et al. “Matterport3D: Learning from RGB-D Data in Indoor Environments”. In: *3DV*. 2017.
- [23] Mark Chavira and Adnan Darwiche. “On Probabilistic Inference by Weighted Model Counting”. In: *Artificial Intelligence* 172.6-7 (2008), pp. 772–799.

- [24] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *ICML*. 2020.
- [25] Yoojung Choi, Antonio Vergari, and Guy Van den Broeck. “Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Modeling”. UCLA. 2020.
- [26] Junyoung Chung et al. “A Recurrent Latent Variable Model for Sequential Data”. In: *NeurIPS*. 2015.
- [27] Nuri Cingillioglu and Alessandra Russo. “DeepLogic: Towards End-to-End Differentiable Logical Reasoning”. In: *AAAI-MAKE*. 2019.
- [28] William W. Cohen, Fan Yang, and Kathryn Mazaitis. “TensorLog: A Probabilistic Database Implemented Using Deep-Learning Infrastructure”. In: *JAIR* 67 (2020), pp. 285–325.
- [29] Michael Collins. “Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms”. In: *EMNLP*. 2002.
- [30] Cristina Cornelio et al. “Learning Where and When to Reason In Neuro-Symbolic Inference”. In: *ICLR*. 2023.
- [31] Daniel Cunnington et al. “The Role of Foundation Models in Neuro-Symbolic Learning and Reasoning”. In: *arXiv* (2024).
- [32] Artur d’Avila Garcez et al. “Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning”. In: *Journal of Applied Logics* 6.4 (2019), pp. 611–632.
- [33] Artur S. d’Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer, 2002.
- [34] Artur S. d’Avila Garcez, Luís C. Lamb, and Dov M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Springer, 2009.
- [35] Adnan Darwiche. “SDDs: A New Canonical Representation of Propositional Knowledge Bases”. In: *ICDT*. 2011.
- [36] Sridhar Dasaratha et al. “DeepPSL: End-to-End Perception and Reasoning”. In: *IJCAI*. 2023.

- [37] Tirtharaj Dash et al. “A Review of Some Techniques for Inclusion of Domain-Knowledge into Deep Neural Networks”. In: *Scientific Reports* 12.1 (2022), p. 1040.
- [38] Peter Dayan et al. “The Helmholtz Machine”. In: *Neural Computation* 7.5 (1995), pp. 889–904.
- [39] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. “ProbLog: A Probabilistic Prolog and Its Application in Link Discovery”. In: *IJCAI*. 2007.
- [40] Matt Deitke et al. “RoboTHOR: An Open Simulation-to-Real Embodied AI Platform”. In: *CVPR*. 2020.
- [41] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. “Lifted Rule Injection for Relation Embeddings”. In: *EMNLP*. 2016.
- [42] Vincent Derkinderen et al. “Semirings for Probabilistic and Neuro-Symbolic Logic Programming”. In: *International Journal of Approximate Reasoning* (2024), p. 109130.
- [43] Charles Dickens. “A Unifying Mathematical Framework for Neural-Symbolic Systems”. PhD thesis. University of California, Santa Cruz, 2024.
- [44] Charles Dickens, Connor Pryor, and Lise Getoor. “Modeling Patterns for Neural-Symbolic Reasoning using Energy-Based Models”. In: *AAAI Spring Symposium on Empowering Machine Learning and Large Language Models with Domain and Commonsense Knowledge*. 2024.
- [45] Charles Dickens et al. “Convex and Bilevel Optimization for Neuro-Symbolic Inference and Learning”. In: *ICML*. 2024.
- [46] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. “Semantic-Based Regularization for Learning and Inference”. In: *Journal of Machine Learning Research* 18 (2017), pp. 1–45.
- [47] Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. “Integrating Prior Knowledge into Deep Learning”. In: *ICMLA*. 2017.
- [48] Chuong Do, Chuan-Sheng Foo, and Andrew Ng. “Efficient Multiple Hyperparameter Learning for Log-Linear Models”. In: *NeurIPS*. 2007.

- [49] Honghua Dong et al. “Neural Logic Machines”. In: *ICLR*. 2019.
- [50] Yilun Du and Igor Mordatch. “Implicit Generation and Modeling with Energy-based Models”. In: *NeurIPS*. 2019.
- [51] Yilun Du et al. “Reduce, Reuse, Recycle: Compositional Generation with Energy-based Diffusion Models and MCMC”. In: *ICML*. 2023.
- [52] Richard Evans and Edward Grefenstette. “Learning Explanatory Rules from Noisy Data”. In: *JAIR* 61 (2018), pp. 1–64.
- [53] Christodoulos A. Floudas and Panos M. Pardalos. *Encyclopedia of Optimization*. Springer Science & Business Media, 2008.
- [54] Samir Yitzhak Gadre et al. “Cows on Pasture: Baselines and Benchmarks for Language-Driven Zero-Shot Object Navigation”. In: *CVPR*. 2023.
- [55] Artur d’Avila Garcez and Luis C. Lamb. “Neurosymbolic AI: The 3rd Wave”. In: *AI Review* 56.11 (2023), pp. 12387–12406.
- [56] Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [57] Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. “Deep Learning with Logical Constraints”. In: *IJCAI*. 2022.
- [58] Eleonora Giunchiglia et al. “ROAD-R: The Autonomous Driving Dataset with Logical Requirements”. In: *Machine Learning* 112.1 (2023), pp. 3261–3291.
- [59] James Gosling, Mike Sheridan, and Patrick Naughton. *The Java™ Programming Language*. <https://www.oracle.com/java/>. 1995.
- [60] Will Grathwohl et al. “Your Classifier is Secretly an Energy-based Model and You Should Treat it Like One”. In: *ICLR*. 2020.
- [61] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *NeurIPS*. 2017.
- [62] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CVPR*. 2016.

- [63] Geoffrey Hinton. “Training Products of Experts by Minimizing Contrastive Divergence”. In: *Neural Computation* 14.8 (2002), pp. 1771–1800.
- [64] Zhiting Hu et al. “Harnessing Deep Neural Networks with Logic Rules”. In: *ACL*. 2016.
- [65] Peter Jung, Giuseppe Marra, and Ondřej Kuželka. “Quantified Neural Markov Logic Networks”. In: *IJAR* 171 (2024), p. 109172.
- [66] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *ICLR*. 2014.
- [67] Thomas Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *ICLR*. 2017.
- [68] Doga Kisa et al. “Probabilistic Sentential Decision Diagrams”. In: *KR*. 2014.
- [69] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. The MIT Press, 2009.
- [70] Emile van Krieken. “Optimisation in Neurosymbolic Learning Systems”. PhD thesis. Radboud University, 2024.
- [71] Emile van Krieken et al. “On the Independence Assumption in Neurosymbolic Learning”. In: *ICML*. 2024.
- [72] Emile van Krieken et al. “ULLER: A Unified Language for Learning and Reasoning”. In: *NeSy*. 2024.
- [73] Luís C. Lamb et al. “Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective”. In: *IJCAI*. 2020.
- [74] Yann LeCun et al. “A Tutorial on Energy-Based Learning”. In: *Predicting Structured Data* 1.0 (2006).
- [75] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *IEEE* 86.11 (1998), pp. 2278–2324.
- [76] Weitang Liu et al. “Energy-based Out-of-distribution Detection”. In: *NeurIPS*. 2020.

- [77] Jaron Maene, Vincent Derkinderen, and Luc De Raedt. “On the Hardness of Probabilistic Neurosymbolic Learning”. In: *arXiv* (2024).
- [78] Jaron Maene and Luc De Raedt. “Soft-Unification in Deep Probabilistic Logic”. In: *NeurIPS*. 2024.
- [79] Arjun Majumdar et al. “ZSON: Zero-Shot Object-Goal Navigation Using Multi-modal Goal Embeddings”. In: *NeurIPS*. 2022.
- [80] Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. “Approximate Inference for Neural Probabilistic Logic Programming”. In: *ICPKRR*. 2021.
- [81] Robin Manhaeve et al. “Neural Probabilistic Logic Programming in DeepProbLog”. In: *AI* 298 (2021), p. 103504.
- [82] Emanuele Marconato, Stefano Teso, and Andrea Passerini. “Neuro-Symbolic Reasoning Shortcuts: Mitigation Strategies and Their Limitations”. In: *NeSy Workshop*. 2023.
- [83] Emanuele Marconato et al. “BEARS Make Neuro-Symbolic Models Aware of Their Reasoning Shortcuts”. In: *UAI*. 2024.
- [84] Emanuele Marconato et al. “Not All Neuro-Symbolic Concepts Are Created Equal: Analysis and Mitigation of Reasoning Shortcuts”. In: *NeurIPS*. 2023.
- [85] Giuseppe Marra. “Bridging Symbolic and Subsymbolic Reasoning with Minimax Entropy Models”. In: *IA* 15.2 (2022), pp. 71–90.
- [86] Giuseppe Marra and Ondřej Kuželka. “Neural Markov Logic Networks”. In: *UAI*. 2021.
- [87] Giuseppe Marra et al. “From Statistical Relational to Neurosymbolic Artificial Intelligence: A Survey”. In: *AI* 328 (2024), p. 104062.
- [88] Giuseppe Marra et al. “Integrating Learning and Reasoning with Deep Logic Models”. In: *ECMLKDD*. 2019.
- [89] Giuseppe Marra et al. “Relational Neural Machines”. In: *ECAI*. 2020.

- [90] Pedro Zuidberg Dos Martires, Luc De Raedt, and Angelika Kimmig. “Declarative Probabilistic Logic Programming in Discrete-Continuous Domains”. In: *AI 337* (2024), p. 104227.
- [91] Lina Mezghani et al. “Memory-Augmented Reinforcement Learning for Image-Goal Navigation”. In: *IROS*. 2022.
- [92] Paul Milgrom and Ilya Segal. “Envelope Theorems for Arbitrary Choice Sets”. In: *Econometrica* 70.2 (2002), pp. 583–601.
- [93] Marvin Minsky. “Logical vs. Analogical or Symbolic vs. Connectionist or Neat vs. Scruffy”. In: *AI Magazine* 12.2 (1991), pp. 34–51.
- [94] Eleonora Misino, Giuseppe Marra, and Emanuele Sansone. “VAEL: Bridging Variational Autoencoders and Probabilistic Logic Programming”. In: *NeurIPS*. 2022.
- [95] Till Mossakowski. “Modular Design Patterns for Neural-Symbolic Integration: Refinement and Combination”. In: *NeSy Workshop* (2022).
- [96] *Neural-Symbolic Learning and Reasoning Workshop at IJCAI*. 2005.
- [97] *International Conference on Neural-Symbolic Learning and Reasoning*. 2024.
- [98] OpenAI. *GPT-4 Technical Report*. Tech. rep. OpenAI, 2024.
- [99] Liangming Pan et al. “Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning”. In: *EMNLP*. 2023.
- [100] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *NeurIPS*. 2019.
- [101] Karl Pearson. “The Problem of the Random Walk”. In: *Nature* 72.1867 (1905), pp. 342–342.
- [102] Fabian Pedregosa. “Hyperparameter Optimization with Approximate Gradient”. In: *ICML*. 2016.
- [103] Connor Pryor et al. “NeuPSL: Neural Probabilistic Soft Logic”. In: *IJCAI*. 2023.

- [104] Connor Pryor et al. “Using Domain Knowledge to Guide Dialog Structure Induction via Neural Probabilistic Soft Logic”. In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. Toronto, Canada, 2023.
- [105] Aravind Rajeswaran et al. “Meta-Learning with Implicit Gradients”. In: *NeurIPS*. 2019.
- [106] Santhosh Kumar Ramakrishnan et al. “Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI”. In: *NeurIPS*. 2021.
- [107] Abhinav Rastogi et al. “Towards Scalable Multi-Domain Conversational Agents: The Schema-Guided Dialogue Dataset”. In: *AAAI*. 2020.
- [108] Tim Rocktäschel and Sebastian Riedel. “End-to-end Differentiable Proving”. In: *NeurIPS*. 2017.
- [109] Guido van Rossum and Python Software Foundation. *Python Programming Language*. <https://www.python.org/>. 1991.
- [110] Ruslan Salakhutdinov and Hugo Larochelle. “Efficient Learning of Deep Boltzmann Machines”. In: *AISTATS*. 2010.
- [111] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80.
- [112] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *ICCV*. 2017.
- [113] Prithviraj Sen et al. “Collective Classification in Network Data”. In: *AI Magazine* 29.3 (2008), pp. 93–106.
- [114] Luciano Serafini and Artur S. d’Avila Garcez. “Learning and Reasoning with Logic Tensor Networks”. In: *AI*IA*. 2016.
- [115] Weiyan Shi, Tiancheng Zhao, and Zhou Yu. “Unsupervised Dialog Structure Learning”. In: *ACL*. 2019.
- [116] Hikaru Shindo et al. “ α ILP: Thinking Visual Scenes as Differentiable Logic Programs”. In: *ML* 112.5 (2023), pp. 1465–1497.

- [117] Karan Sikka et al. *Deep Adaptive Semantic Logic (DASL): Compiling Declarative Knowledge into Deep Neural Networks*. Tech. rep. SRI International, 2020.
- [118] Gurkirt Singh et al. “ROAD: The Road Event Awareness Dataset for Autonomous Driving”. In: *IEEE TPAMI* 45 (2021), pp. 1036–1054.
- [119] Gustav Sourek et al. “Lifted Relational Neural Networks: Efficient Learning of Latent Relational Structures”. In: *JAIR* 62 (2018), pp. 69–100.
- [120] Sriram Srinivasan et al. “A Taxonomy of Weight Learning Methods for Statistical Relational Learning”. In: *Machine Learning* (2021).
- [121] Aarohi Srivastava et al. “Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models”. In: *arXiv* (2022).
- [122] Mihaela Cătălina Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. “Exploiting T-norms for Deep Learning in Autonomous Driving”. In: *NeSy*. 2023.
- [123] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.
- [124] Richard Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *NeurIPS*. 1999.
- [125] Geoffrey G. Towell and Jude W. Shavlik. “Knowledge-Based Artificial Neural Networks”. In: *AI* 70.1-2 (1994), pp. 119–165.
- [126] Son N. Tran and Artur S. d’Avila Garcez. “Deep Logic Networks: Inserting and Extracting Knowledge From Deep Belief Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 29.2 (2018), pp. 246–258.
- [127] Emile van Krieken, Erman Acar, and Frank van Harmelen. “Analyzing Differentiable Fuzzy Logic Operators”. In: *Artificial Intelligence (AI)* 302 (2022), p. 103602.
- [128] Emile van Krieken et al. “A-NeSI: A Scalable Approximate Method for Probabilistic Neurosymbolic Inference”. In: *NeurIPS*. 2023.
- [129] Ashish Vaswani et al. In: *Attention Is All You Need*. Vol. 30. NIPS, 2017, pp. 5998–6008.

- [130] Nguyen Xuan Vinh, Julien Epps, and James Bailey. “Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance”. In: *Journal of Machine Learning Research* 11 (2010), pp. 2837–2854.
- [131] Marin Vlastelica et al. “Differentiation of Blackbox Combinatorial Solvers”. In: *ICLR*. 2020.
- [132] Po-Wei Wang et al. “Satnet: Bridging Deep Learning and Logical Reasoning Using a Differentiable Satisfiability Solver”. In: *ICML*. 2019.
- [133] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *NeurIPS*. 2022.
- [134] Max Welling and Yee W Teh. “Bayesian Learning via Stochastic Gradient Langevin Dynamics”. In: *ICML*. 2011.
- [135] Ronald Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Machine Learning* 8 (1992), pp. 229–256.
- [136] Thomas Winters et al. “DeepStochLog: Neural Stochastic Logic Programming”. In: *AAAI*. 2022.
- [137] Felix Wu et al. “Simplifying Graph Convolutional Networks”. In: *ICML*. 2019.
- [138] Jingyi Xu et al. “A Semantic Loss Function for Deep Learning with Symbolic Knowledge”. In: *ICML*. 2018.
- [139] B. Yamauchi. “A Frontier-Based Approach for Autonomous Exploration”. In: *CIRA*. 1997.
- [140] Fan Yang, Zhilin Yang, and William W. Cohen. “Differentiable Learning of Logical Rules for Knowledge Base Reasoning”. In: *NeurIPS*. 2017.
- [141] Zhun Yang, Adam Ishay, and Joohyung Lee. “NeurASP: Embracing Neural Networks into Answer Set Programming”. In: *IJCAI*. 2020.
- [142] Dongran Yu et al. “A Survey on Neural-Symbolic Learning Systems”. In: *NN* 166 (2023), pp. 105–126.

- [143] Jing Zhang et al. “Neural, Symbolic, and Neural-Symbolic Reasoning on Knowledge Graphs”. In: *AI Open* 2 (2021), pp. 14–35.
- [144] Junbo Zhao, Michael Mathieu, and Yann LeCun. “Energy-based Generative Adversarial Networks”. In: *ICLR*. 2017.
- [145] Kaiwen Zhou et al. “Esc: Exploration with Soft Commonsense Constraints for Zero-Shot Object Navigation”. In: *ICML*. 2023.

Appendix A

Extended Model Details

This appendix serves to provide additional details and supplementary information supporting the experimental evaluation and methodologies presented in this dissertation. The appendix is organized as follows:

- **NeuPSL Symbolic Constraints** (Appendix [A.1](#)): This section provides the base set of NeuPSL symbolic constraints used across each dataset featured in the experimental evaluation (Section [7.1](#)).

A.1 NeuPSL Symbolic Constraints

This section provides a foundational overview of the NeuPSL symbolic constraints employed across the datasets used in the experimental evaluation (Section [7.1](#)). The focus here is on presenting the base constraints in the NeuPSL models for each experimental setting. While this overview outlines the core structure of the models, variations introduced in specific empirical scenarios (Chapter [7](#)) or fine-grained details regarding the exact NeuPSL syntax are excluded. The primary aim of this section is to offer a clear understanding of the baseline rule sets utilized for each dataset, serving as a reference point for the experiments conducted throughout the dissertation. This section is organized as follows: MNIST-Add1 (Section [A.1.1](#)), MNIST-Add2 (Section [A.1.2](#)), Visual Sudoku (Section [A.1.3](#)), Pathfinding (Section [A.1.4](#)), Citation Network (Section [A.1.5](#)), RoadR (Section [A.1.6](#)), Zero-Shot

Object Navigation (Section A.1.7), Dialog Structure Induction (Section A.1.8), Synthetic Mixture of Experts (Section A.1.9), and Logic Deduction (Section A.1.10).

Figure A.1: NeuPSL MNIST-Add1 Symbolic Model

```
# Digit Sums
w1 : NEURAL(Img1, X) ∧ NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → SUM(Img1, Img2, Z)
w2 : ¬NEURAL(Img1, X) ∧ NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)
w3 : NEURAL(Img1, X) ∧ ¬NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)

# Digit Constraints
w4 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}
w5 : NEURAL(Img2, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}

# Simplex Constraints
SUM(Img1, Img2, +Z) = 1.
```

A.1.1 MNIST-Add1

The NeuPSL model for the *MNIST-Add1* experiments incorporates symbolic constraints depicted in Figure A.1. The symbolic model includes the following predicates:

- **Neural(Img, X)** The NEURAL predicate is the class probability for each image as inferred by the neural network. **Img** is MNIST image identifier and **X** is a digit class that the image may represent.
- **DigitSum(X, Y, Z)** The DIGITSUM predicate determines if two digits (**X** and **Y**) sum to a number (**Z**). For example, DIGITSUM(4, 5, 9) would return 1 as 4 added to 5 is 9. Conversely, DIGITSUM(2, 2, 5) would return 0 as 2 added to 2 is not 5.
- **Sum(Img1, Img2, Z)** The SUM predicate is the probability that the digits represented in the images identified by arguments **Img1** and **Img2** add up to the number identified by the argument **Z**. This predicate instantiates decision variables, i.e., variables from

this predicate are not fixed during inference and learning as described in the NeSy EBM, NeuPSL, and Inference and Learning sections.

- **PossibleDigits(X, Z)** The POSSIBLEDIGITS predicate determines if a digit (X) can be included in a sum that equals a number (Z). For example, POSSIBLEDIGITS(9, 0) would return 0 as no positive digit when added to 9 will equal 0. Conversely, POSSIBLEDIGITS(9, 17) would return 1 as 8 added to 9 equals 17.

The *Digit Sums* constraints represents the summation of the two images **Img1** and **Img2**, i.e., if the neural model labels the image id **Img1** as digit X and **Img2** as Y and the digits X and Y sum to Z then the sum of the images must be Z .

The *Digit Constraints* constraints restrict the possible values of the SUM predicate based on the neural model’s prediction. For instance, if the neural model predicts that the digit label for image **Img1** is 1, then the sum that **Img1** is involved in cannot be any less than 1 or greater than 10.

A.1.2 MNIST-Add2

The NeuPSL model for the *MNIST-Add2* experiment incorporates symbolic constraints depicted in Figure A.2. The symbolic model includes the following predicates:

- **Neural(**Img**, X)** The NEURAL predicate is the class probability for each image as inferred by the neural network. **Img** is MNIST image identifier and X is a digit class that the image may represent.
- **DigitSum(X, Y, Z)** The DIGITSUM predicate determines if two digits (X and Y) sum to a number (Z). For example, DIGITSUM(4, 5, 9) would return 1 as 4 added to 5 is 9. Conversely, DIGITSUM(2, 2, 5) would return 0 as 2 added to 2 is not 5.
- **Sum(**Img1**, **Img2**, **Img3**, **Img4**, Z)** The SUM predicate is the probability that the numbers represented in the images identified by arguments (**Img1**, **Img2**) and (**Img3**, **Img4**) add up to the number identified by the argument Z . This predicate instantiates decision variables, i.e., variables from this predicate are not fixed during inference and

Figure A.2: NeuPSL MNIST-Add2 Symbolic Model

```

# Tens Digit Sums
w1 : NEURAL(Img1, X) ∧ NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → IMAGEDIGITSUM(Img1, Img3, Z)
w2 : ¬NEURAL(Img1, X) ∧ NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img1, Img3, Z)
w3 : NEURAL(Img1, X) ∧ ¬NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img1, Img3, Z)

# Ones Digit Sums
w4 : NEURAL(Img2, X) ∧ NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → IMAGEDIGITSUM(Img2, Img4, Z)
w5 : ¬NEURAL(Img2, X) ∧ NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img2, Img4, Z)
w6 : NEURAL(Img2, X) ∧ ¬NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img2, Img4, Z)

# Place Digit Sums
IMAGEDIGITSUM(Img1, Img3, Z10) ∧ IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ SUM(Img1, Img2, Img3, Img4, Z)
¬IMAGEDIGITSUM(Img1, Img3, Z10) ∧ IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ ¬SUM(Img1, Img2, Img3, Img4, Z)
IMAGEDIGITSUM(Img1, Img3, Z10) ∧ ¬IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ ¬SUM(Img1, Img2, Img3, Img4, Z)

# Tens Digit Constraints
w7 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENS DIGITS(X, Z)}
w8 : NEURAL(Img3, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENS DIGITS(X, Z)}

# Ones Digit Constraints
w9 : NEURAL(Img2, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONES DIGITS(X, Z)}
w10 : NEURAL(Img4, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONES DIGITS(X, Z)}

# Digit Sum Constraints
w11 : NEURAL(Img1, +X) >= IMAGEDIGITSUM(Img1, Img3, Z){X : POSSIBLEDIGITS(X, Z)}
w12 : NEURAL(Img3, +X) >= IMAGEDIGITSUM(Img1, Img3, Z){X : POSSIBLEDIGITS(X, Z)}
w13 : NEURAL(Img2, +X) >= IMAGEDIGITSUM(Img2, Img4, Z){X : POSSIBLEDIGITS(X, Z)}
w14 : NEURAL(Img4, +X) >= IMAGEDIGITSUM(Img2, Img4, Z){X : POSSIBLEDIGITS(X, Z)}

# Number Sum Constraints
IMAGEDIGITSUM(Img1, Img3, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENS SUMS(X, Z)}
IMAGEDIGITSUM(Img2, Img4, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONES SUMS(X, Z)}

# Simplex Constraints
SUM(Img1, Img2, Img3, Img4, +X) = 1.
IMAGEDIGITSUM(Img1, Img2, +X) = 1.

```

learning as described in the NeSy EBM, NeuPSL, and Inference and Learning sections.

- **PossibleTenDigits(X, Z)** POSSIBLETENDIGITS takes a 0 or 1 value representing whether the digit identified by the argument X is possible when it is in the tens place of a number involved in a sum that totals to the number identified by the argument Z . For instance POSSIBLETENDIGITS(9, 70) = 0 as no positive number added to a number with a 9 in the tens place, e.g., 92, equals 70, while POSSIBLETENDIGITS(9, 170) = 1 as 78 added to 92 is 170.
- **PossibleOnesDigits(X, Z)** POSSIBLEONESDIGITS takes a 0 or 1 value representing whether the digit identified by the argument X is possible when it is in the ones place of a number involved in a sum that totals to the number identified by the argument Z . For instance POSSIBLEONESDIGITS(9, 7) = 0 as no positive number added to a number with a 9 in the ones place, e.g., 9, equals 7 while POSSIBLEONESDIGITS(9, 170) = 1 as 71 added to 99 is 170.
- **ImageDigitSum($Img1, Img2, Z$)** The IMAGEDIGITSUM predicate is the probability that the digits represented in the images specified by $Img1$ and $Img2$ will sum up to the number indicated by the argument Z . These variables are considered latent in the NeuPSL model as there are no truth labels for sums of images in the ones or tens places.
- **PlaceNumberSum($Z10, Z1, Z$)** The PLACENUMBERSUM predicate takes a 0 or 1 value representing whether the sum of the numbers $Z10$ and $Z1$, where $Z10$ is the sum of digits in the tens place and $Z1$ is the sum of digits in the one place, adds up to the number Z . For instance PLACENUMBERSUM(1, 15, 25) is 1 as $1 \cdot 10 + 15 = 25$.

The *Tens Digit Sums* and *Ones Digit Sums* constraints compute the sum of two images in the same manner as the *Digit Sums* constraints in the MNIST-Add1 model. The sum of the digits is captured by the latent variables instantiated by the predicate IMAGEDIGITSUM.

The *Place Digit Sums* constraints use the value of the `IMAGEDIGITSUM` variables to infer the sum of the images. More specifically, if the `IMAGEDIGITSUM` of the images in the tens place, `Img1` and `Img3`), is `Z10`, and the `IMAGEDIGITSUM` of the images in the ones place, `Img2` and `Img4`) is `Z1`, and if according to `PLACENUMBERSUM` the sum of the numbers `Z10` and `Z1` is `Z`, then the `SUM` of the images must be `Z`. Notice that these constraints are hard constraints as it is always possible and desirable to find values of the `IMAGEDIGITSUM` and `SUM` variables that satisfy these relations.

The *Tens Digit Constraint* constraints restrict the possible values of the `SUM` predicate based on the neural model’s prediction for the digit in the tens place of a number. For instance, if the neural model predicts that the digit label for the image `Img1` is 1 and `Img1` is in the tens place of a number, then the sum that `Img1` is involved in cannot be any less than 10 or greater than 118.

The *Ones Digit Constraint* constraints restrict the possible values of the `SUM` predicate based on the neural model’s prediction for the digit in the ones place of a number. For instance, if the neural model predicts that the digit label for the image `Img2` is 5 and `Img2` is in the one place of a number, then the sum that `Img2` is involved in cannot be any less than 5 or greater than 194.

The *Number Sum Constraint* constraints limit the values that `IMAGEDIGITSUM` and `SUM` can take using constraints representing the possible sums in the tens and ones place. For instance, if the `IMAGEDIGITSUM` of two images, `Img1` and `Img3`, both in the tens place of two numbers being added, is 17, then the `SUM` cannot be less than 170 or greater than 188. Furthermore, if the `IMAGEDIGITSUM` of two images, `Img2` and `Img4`, both in the ones place of two numbers being added, is 17, then the `SUM` cannot be less than 17 or greater than 197, and must have a 7 in the ones place.

A.1.3 Visual Sudoku

The NeuPSL model for the *visual sudoku* experiments incorporates symbolic constraints depicted in Figure A.3. The symbolic model includes the following predicates:

- **Neural(Puzzle, X, Y, Number)** The `NEURAL` predicate contains the output class prob-

Figure A.3: NeuPSL Visual Sudoku Symbolic Model

```

# Row Constraint
NEURAL(Puzzle, +X, Y, Number) = 1.

# Column Constraint
NEURAL(Puzzle, X, +Y, Number) = 1.

# Block Constraint
NEURAL(Puzzle, "0", "0", Number) + NEURAL(Puzzle, "0", "1", Number)
    + NEURAL(Puzzle, "1", "0", Number) + NEURAL(Puzzle, "1", "1", Number) = 1.
NEURAL(Puzzle, "2", "0", Number) + NEURAL(Puzzle, "2", "1", Number)
    + NEURAL(Puzzle, "3", "0", Number) + NEURAL(Puzzle, "3", "1", Number) = 1.
NEURAL(Puzzle, "0", "2", Number) + NEURAL(Puzzle, "0", "3", Number)
    + NEURAL(Puzzle, "1", "2", Number) + NEURAL(Puzzle, "1", "3", Number) = 1.
NEURAL(Puzzle, "2", "2", Number) + NEURAL(Puzzle, "2", "3", Number)
    + NEURAL(Puzzle, "3", "2", Number) + NEURAL(Puzzle, "3", "3", Number) = 1.

# Pin First Column
w2 : FIRSTPUZZLE(Puzzle, X, Y) - NEURAL(Puzzle, X, Y) = 0.0

```

ability for each digit image inferred by the neural network. **Puzzle** is sudoku puzzle's identifier, **X** and **Y** represent the location of image in the puzzle, and **Number** is a digit that image may represent.

- **Digit(Puzzle, X, Y, Number)** The **DIGIT** predicate has the same arguments as the **NEURAL** predicate, representing PSL's digit prediction on the image.
- **FirstPuzzle, X, Y(Puzzle)** The **FIRSTPUZZLE** predicate pins the values for the first row of the first puzzle to an arbitrary assignment. This is used to force the neural model to learn the correct label representation for easier evaluation.

The *Row Constraint*, *Column Constraint*, and *Block Constraint* constraints encode the standard Sudoku constraints into constraints. These constraints restrict multiple instances of a digit from appearing in a row, column, or block, respectively.

The *Pin First Column* constraints are used to assign arbitrary classes to the first row of a Sudoku puzzle. The first row of the first correct puzzle from the training set is used to determine this arbitrary label assignment. By assigning the first row to arbitrary classes, the neural model is provided a starting point for differentiating between the different classes and makes the final evaluation easier.

Figure A.4: NeuPSL Pathfinding Symbolic Model

```
# Neural Cost
w1 : COST(X, Y) = PATH(X, Y, X1, Y1)

# Sink and Source Nodes
START(X, Y) → PATH(" - 1", " - 1", X, Y).
END(X, Y) → PATH(X, Y, " - 1", " - 1", ).

# Path Flow Constraints
PATH(X, Y, +OutX, +OutY) <= 1.
PATH(+InX, +InY, X, Y) <= 1.
```

A.1.4 Pathfinding

The NeuPSL model for the *pathfinding* experiments incorporates symbolic constraints depicted in Figure A.4. The symbolic model includes the following predicates:

- **Path**(X1, Y1, X2, Y2) The PATH predicate represents the existence of a path between two nodes. X1 and Y1 denote the source node coordinates, while X2 and Y2 denote the destination node coordinates. This predicate is true if a valid path exists between the nodes.
- **Cost**(X, Y) The COST predicate captures the traversal cost from the node at (X, Y). This cost is inferred from the neural network’s predictions and integrated into the symbolic reasoning process for optimization.

- **Start**(X, Y) The START predicate identifies the starting node in the pathfinding problem. It is true if the node at (X, Y) is the source of the path.
- **End**(X, Y) The END predicate identifies the destination node. It is true if the node at (X, Y) is the target or sink for the path.

The *Neural Cost Constraint* ensures that neural predictions are used to guide the cost of traversal, integrating the neural model’s capabilities into the symbolic reasoning process. The *Source and Sink Constraints* enforce boundary conditions for the paths, ensuring paths begin and end at the correct nodes. Finally, the *Path Flow Constraints* enforce the path’s validity by restricting nodes to only participate in one incoming and one outgoing path, maintaining flow consistency.

Figure A.5: NeuPSL Citation Network Symbolic Model

```
# L2 Loss
w1 : NEURAL(Paper, Label) = CATEGORY(Paper, Label)

# Label Propagation
w2 : LINK(Paper1, Paper2) ∧ CATEGORY(Paper1, Label) → CATEGORY(Paper2, Label)

# Simplex Constraints
CATEGORY(Paper, +Label) = 1.
```

A.1.5 Citation Network

The NeuPSL model for the citation network experiments incorporates symbolic constraints depicted in Figure A.5. The symbolic model includes the following predicates:

- **Neural**($\text{Paper}, \text{Label}$) The NEURAL predicate contains the output class probability for each paper as inferred by the neural network. **Paper** is the identifier and **Label** is the category it can take.
- **Category**($\text{Paper}, \text{Label}$) The CATEGORY predicate has the same arguments as the NEURAL predicate and represents PSL’s label prediction on the paper.

- **Link(Paper1, Paper2)** The LINK predicate denotes whether two papers share a citation link.

The *Label Propagation* rule propagates node labels to neighbors. In this sense, it encodes the idea that papers sharing a citation link are likely to have the same underlying label category.

Figure A.6: NeuPSL RoadR Object Detection Symbolic Model

```
# Logical Constraints for Class Co-occurrence
1.0 : NEURAL(FrameID, BoundingBoxID, Class1)  $\wedge$   $\neg$ CoOccurrence(Class1, Class2)
       $\rightarrow$   $\neg$ NEURAL(FrameID, BoundingBoxID, Class2)

# One Agent Constraint
1.0 : NEURAL(FrameID, BoundingBoxID, +Class) = 1    {Class : AGENT(Class)}

# At Least One Action Constraint
1.0 : NEURAL(FrameID, BoundingBoxID, +Class)  $\geq$  1    {Class : ACTION(Class)}

# At Least One Location Constraint
1.0 : NEURAL(FrameID, BoundingBoxID, +Class) + NEURAL(FrameID, BoundingBoxID, traffic light)
      + NEURAL(FrameID, BoundingBoxID, other traffic light)  $\geq$  1    {Class : LOCATION(Class)}
```

A.1.6 RoadR

The NeuPSL model for the Road-R experiments incorporates symbolic constraints depicted in Figure A.6. The symbolic model includes the following predicates:

- **Neural(FrameID, BoundingBoxID, Class)**: This predicate represents the neural network's predicted probability that the bounding box identified by **BoundingBoxID** in frame **FrameID** corresponds to the class **Class**.
- **CoOccurrence(Class1, Class2)**: Specifies whether two object classes, **Class1** and **Class2**, are allowed to co-occur within a single frame. There are 861 co-occurrence binary values.

- **Agent(Class)**: Indicates whether the class **Class** is an agent.
- **Action(Class)**: Indicates whether the class **Class** is an action.
- **Location(Class)**: Indicates whether the class **Class** is a location.

The *Logical Constraints for Class Co-occurrence* ensure that object classes which are mutually exclusive due to semantic or domain requirements cannot co-occur. For example, a traffic light cannot simultaneously be classified as both green and red, so the model must enforce that at least one of these states is false for any given detection. The *One Agent Constraint* ensures that each bounding box is associated with exactly one agent. The *At Least One Action Constraint* enforces that every bounding box is associated with at least one action. The *At Least One Location Constraint* guarantees that every bounding box has at least one location, unless the agent is a traffic light or other traffic light (these do not need to have a location).

Figure A.7: NeuPSL Zero Shot Object Navigation Symbolic Model

```
# Object Reasoning
w1 : IsCOOCCUR(Goal, Object) ∧ IsNEAROBJ(Frontier, Object) → CHOOSEFRONTIER(Frontier)
w2 : !IsCOOCCUR(Goal, Object) ∧ IsNEAROBJ(Frontier, Object) → !CHOOSEFRONTIER(Frontier)

# Room Reasoning
w3 : IsCOOCCUR(Goal, Room) ∧ IsNEAROBJ(Frontier, Object) → CHOOSEFRONTIER(Frontier)
w4 : !IsCOOCCUR(Goal, Room) ∧ IsNEAROBJ(Frontier, Object) → !CHOOSEFRONTIER(Frontier)

# Distant Constraint
w5 : SHORTDIST(Frontier) → CHOOSEFRONTIER(Frontier)

# Simplex Constraints
CHOOSEFRONTIER(+Frontier) = 1.
```

A.1.7 Zero-Shot Object Navigation

The NeuPSL model for the zero-shot object navigation experiments incorporates symbolic constraints depicted in Figure A.7. The symbolic model includes the following predicates:

- **IsCooccur**(Goal, Object) The ISCOCCUR predicate contains the co-occurrence score for each object and goal pair inferred by the neural network. **Goal** is the goal identifier and **Object** is the object identifier.
- **IsCooccur**(Goal, Room) The ISCOCCUR predicate contains the co-occurrence score for each room and goal pair inferred by the neural network. **Goal** is the goal identifier and **Room** is the room identifier.
- **ChooseFrontier**(Frontier) The CHOOSEFRONTIER predicate contains the score for each frontier that can be taken.
- **ShortDist**(Frontier) The SHORTDIST predicate contains the distance to each frontier.

The *Object and Room Reasoning* rules assign weight to the frontier by leveraging the LLM’s reasoning about the likelihood of specific objects and rooms being associated with the goal. In essence, this approach guides the frontier based on the LLM’s assessment of the relevance of each option.

A.1.8 Dialog Structure Induction

The NeuPSL model for the dialog structure induction experiments incorporates symbolic constraints depicted in Figure A.8 for the SGD data settings and Figure A.9 for the MultiWoZ data settings.

SGD: The SGD symbolic model includes the following predicates:

- **State**(Utt, Class)

The STATE continuous valued predicate is the probability that an utterance, identified

Figure A.8: NeuPSL SGD Dialog Structure Induction Symbolic Model

<p># <i>Token Constraint</i></p> <p>$w_1 : \text{HASWORD}(\text{Utt}, \text{Class}) \rightarrow \text{STATE}(\text{Utt}, \text{Class})$</p>

by the argument **Utt**, belongs to a dialog state, identified by the argument **Class**. For instance, the utterance *hello world !* for the *greet* dialog state would create a predicate with a value between zero and one, i.e., $\text{STATE}(\text{hello world !}, \text{greet}) = 0.7$.

- **HasWord(Utt, Class)**

The **HASWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for a particular class, identified by the argument **Class**. For instance if a known token associated with the *greet* class is *hello*, then the utterance *hello world !* would create a predicate with value one, i.e. $\text{HASWORD}(\text{hello world !}, \text{greet}) = 1$.

The *Token Constraint* encodes the prior knowledge that utterances' are likely to belong to dialog states when an utterance contains tokens representing that state. For example, if a known token associated with the *greet* class is *hello*, then the utterance *hello world !* is likely to belong to the *greet* state.

MultiWoZ: The MultiWoZ symbolic model includes the following predicates:

- **State(Utt, Class)**

The **STATE** continuous valued predicate is the probability that an utterance, identified by the argument **Utt**, belongs to a dialog state, identified by the argument **Class**. For instance, the utterance *hello world !* for the *greet* dialog state would create a predicate with a value between zero and one, i.e., $\text{STATE}(\text{hello world !}, \text{greet}) = 0.7$.

- **FirstUtt(Utt)**

The **FIRSTUTT** binary predicate indicates if an utterance, identified by the argument **Utt**, is the first utterance in a dialog.

Figure A.9: NeuPSL MultiWoZ Dialog Structure Induction Symbolic Model

<pre> # Dialog Start w1 : ¬FIRSTUTT(Utt) → ¬STATE(Utt, greet) w2 : FIRSTUTT(Utt) ∧ HASGREETWORD(Utt) → STATE(Utt, greet) w3 : FIRSTUTT(Utt) ∧ ¬HASGREETWORD(Utt) → STATE(Utt, init_request) # Dialog Middle w4 : PREVUTT(Utt1, Utt2) ∧ STATE(Utt2, greet) → STATE(Utt1, init_request) w5 : PREVUTT(Utt1, Utt2) ∧ ¬STATE(Utt2, greet) → ¬STATE(Utt1, init_request) w6 : PREVUTT(Utt1, Utt2) ∧ STATE(Utt2, init_request) → STATE(Utt1, second_request) w7 : PREVUTT(Utt1, Utt2) ∧ STATE(Utt2, second_request) ∧ HASINFOQUESTIONWORD(Utt1) → STATE(Utt1, info_question) w8 : PREVUTT(Utt1, Utt2) ∧ STATE(Utt2, second_request) ∧ HASLOTQUESTIONWORD(Utt1) → STATE(Utt1, slot_question) w9 : PREVUTT(Utt1, Utt2) ∧ STATE(Utt2, end) ∧ HASCANCELWORD(Utt1) → STATE(Utt1, cancel) # Dialog End w10 : LASTUTT(Utt) ∧ HASENDWORD(Utt) → STATE(Utt, end) w11 : LASTUTT(Utt) ∧ HASACCEPTWORD(Utt) → STATE(Utt, accept) w12 : LASTUTT(Utt) ∧ HASINSISTWORD(Utt) → STATE(Utt, insist) </pre>

- **LastUtt(Utt)**

The LASTUTT binary predicate indicates if an utterance, identified by the argument Utt, is the last utterance in a dialog.

- **PrevUtt(Utt1, Utt2)**

The PREVUTT binary predicate indicates if an utterance, identified by the argument Utt2, is the previous utterance in a dialog of another utterance, identified by the argument Utt1.

- **HasGreetWord(Utt)**

The HASGREETWORD binary predicate indicates if an utterance, identified by the

argument **Utt**, contains a known token for the greet class. The list of known greet words is [*hello*, *hi*].

- **HasInfoQuestionWord(Utt)**

The **HASINFOQUESTIONWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the info question class. The list of known info question words is [*address*, *phone*].

- **HasSlotQuestionWord(Utt)**

The **HASSLOTQUESTIONWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the slot question class. The list of known slot question words is [*what*, *?*].

- **HasInsistWord(Utt)**

The **HASINSISTWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the insist class. The list of known insist words is [*sure*, *no*].

- **HasCancelWord(Utt)**

The **HASCANCELWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the cancel class. The list of known cancel words is [*no*].

- **HasAcceptWord(Utt)**

The **HASACCEPTWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the accept class. The list of known accept words is [*yes*, *great*].

- **HasEndWord(Utt)**

The **HASENDWORD** binary predicate indicates if an utterance, identified by the argument **Utt**, contains a known token for the end class. The list of known end words is [*thank*, *thanks*].

The *Dialog Start* constraints take advantage of the inherent structure built into the beginning of a dialog. 1) If the first turn utterance does not contain a known greet word, then it does not belong to the *greet* state. 2) If the first turn utterance contains a known greet word, then it belongs to the *greet* state. 3) If the first turn utterance does not contain a known greet word, then it belongs to the *initial request* state.

The *Dialog Middle* constraints exploit the temporal dependencies within the middle of a dialog. 1) If the previous utterance belongs to the *greet* state, then the current utterance belongs to the *initial request* state. 2) If the previous utterance does not belong to the *greet* state, then the current utterance does not belong to the *initial request* state. 3) If the previous utterance belongs to the *initial request* state, then the current utterance belongs to the *second request* state. 4) If the previous utterance belongs to the *second request* state and it has a known info question token, then the current utterance belongs to the *info question* state. 5) If the previous utterance belongs to the *second request* state and it has a known slot question token, then the current utterance belongs to the *slot question* state. 4) If the previous utterance belongs to the *end* state and it has a known cancel token, then the current utterance belongs to the *cancel* state.

The *Dialog End* constraints take advantage of the inherent structure built into the end of a dialog. 1) If the last turn utterance contains a known end word, then it belongs to the *end* state. 2) If the last turn utterance contains a known accept word, then it belongs to the *accept* state. 3) If the last turn utterance contains a known insist word, then it belongs to the *insist* state.

A.1.9 Synthetic Mixture of Experts

The NeuPSL model for the synthetic mixture of experts experiments incorporates symbolic constraints depicted in Figure A.10. The symbolic model includes the following predicates:

- **Neural(Paper, Label)** The NEURAL predicate contains the output class probability for each paper as inferred by a baseline neural network. **Paper** is the identifier and **Label** is the category it can take.

Figure A.10: NeuPSL Synthetic Mixture of Experts Symbolic Model

```
# L2 Loss
w1 : NEURAL(Paper, Label) = CATEGORY(Paper, Label)

# Label Propagation
w2 : LINK(Paper1, Paper2) ∧ CATEGORY(Paper1, Label) → CATEGORY(Paper2, Label)

# Simplex Constraints
CATEGORY(Paper, +Label) = 1.
```

- **CATEGORY(Paper, Label)** The CATEGORY predicate has the same arguments as the NEURAL predicate and represents PSL’s label prediction on the paper.
- **LINK(Paper1, Paper2)** The LINK predicate denotes whether two papers share a citation link.

The *Label Propagation* rule propagates node labels to neighbors. In this sense, it encodes the idea that papers sharing a citation link are likely to have the same underlying label category.

A.1.10 Logic Deduction

The NeuPSL model for logic deduction experiments incorporates symbolic constraints created using a LLM, which was prompted with context similarly depicted in Figure A.11.

Figure A.11: NeuPSL Logic Deduction Prompt for Generating Symbolic Model

Task Description:

You are given a problem description. The task is to parse the problem as a logical program, defining the Domain, Predicates, Targets, Rules, and Query. Use plain text formatting with no bullets.

Context:

The following paragraphs each describe a set of three objects arranged in a fixed order. The statements are logically consistent within each paragraph. In an antique car show, there are three vehicles: a station wagon, a convertible, and a minivan. The station wagon is the oldest. The minivan is newer than the convertible.

Options:

- A) The station wagon is the second-newest.
- B) The convertible is the second-newest.
- C) The minivan is the second-newest.

Map Variables:

```
[oldest = x_, second-oldest = _x_, third-oldest = __x]
[newest = __x, second-newest = _x_, third-newest = x_]
```

Domain:

x_, _x_, __x

Predicates:

Order(Vehicle, OrderIndex)

Targets:

```
{
["station-wagon", "convertible", "minivan"],
["x_", "_x_", "__x"]
}
```

Rules:

```
// The station wagon is the oldest: x_
Order("station-wagon", "x_") = 1
// The minivan is newer than the convertible: minivan > convertible
Order("convertible", OrderIndexConvertible) & Order("minivan", OrderIndexMinivan) ->
(OrderIndexConvertible < OrderIndexMinivan)
// Each vehicle has one order index.
Order(Vehicle, "x_") + Order(Vehicle, "_x_") + Order(Vehicle, "__x") = 1
// Each order index is assigned to one vehicle.
Order("station-wagon", OrderIndex) + Order("minivan", OrderIndex) + Order("convertible",
OrderIndex) = 1
```

Query:

```
// The station wagon is the second-newest: _x_
Order("station-wagon", "_x_") = 1
// The convertible is the second-newest: _x_
Order("convertible", "_x_") = 1
// The minivan is the second-newest: _x_
Order("minivan", "_x_") = 1
```